Bio-inspired Neuromorphic Computing Using Memristor Crossbar Networks

by

YeonJoo Jeong

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctorate of Philosophy
(Electrical Engineering)
in the University of Michigan
2018

Doctoral Committee:

Professor Wei D. Lu, Chair
Assistant Professor Ronald G. Dreslinski
Professor L. Jay Guo
Associate Professor Zhengya Zhang

YeonJoo Jeong

yjjeong@umich.edu

ORCID iD: 0000-0001-5855-5066

Dedication

To The almighty God, my wife Kyung Mi, daughter Leah, my family and friends

Acknowledgements

I would like to express my deepest appreciation for all the supports I've been given during my course of study at the University of Michigan. Enormous gratitude is extended to my advisor, Prof. Wei D. Lu, who has given me this grateful opportunity and has guided me with his immense knowledge and inspiring comments. This work would not have been possible without his patience, encouragement, and understanding.

I would also like to thank my committee members, Prof. L. Jay Guo, Prof. Zhengya Zhang, and Prof. Ronald G. Dreslinski for their useful discussions sharing their knowledge and time, which enriches this work from diverse perspectives.

My sincere thanks goes to my parents, family, and friends for their endless support throughout this PhD program and my life. A special thank extends to my wife Kyung Mi Lee for her unconditional love, cheering encouragement, and all the dedication to raise lovely daughter Leah.

I would like to express my appreciation to past and current group members: Dr. Mohammed A. Zidan, Dr. Jihang Lee, Dr. Xiaojian Zhu, Dr. Sungho Kim, Dr. Chao Du, Dr. Wen Ma, Dr. Patrick Sheridan, Fuxi Cai, Jong Hoon Shin, Seunghwan Lee, John Moon, and HeeWoo Kim, each of whom a brilliant individual. In particular, I would like to thank Dr. Mohammed A. Zidan and Dr. Jihang Lee, who have been actively co-worked with me for several successful projects with their helpful discussions and timely assistances.

I would also like to thank the Lurie Nanofabrication Facility (LNF) staff (and former staff) at the University of Michigan for their technical support: Dr. Pilar Herrera-Fierro, Dr.

Sandrine Martin, Matt Oonk, Greg Allion, Tony Sebastian, Vishva Ray, David Sebastian, Nadine Wang, Owen Kevin, Wright Shawn, and Patel Caroline.

Table of Contents

List of Figures

Abstract

Bio-inspired neuromorphic computing systems built with emerging devices such as memristors have become an active research field. Experimental demonstrations at the network-level have suggested memristor-based neuromorphic systems as a promising candidate to overcome the von-Neumann bottleneck in future computing applications. As a hardware system that offers co-location of memory and data processing, memristor-based networks represent an efficient computing platform with minimal data transfer and high parallelism. Furthermore, active utilization of the dynamic processes during resistive switching in memristors can help realize more faithful emulation of biological device and network behaviors, with the potential to process dynamic temporal inputs efficiently.

In this thesis, I present experimental demonstrations of neuromorphic systems using fabricated memristor arrays as well as network-level simulation results. Models of resistive switching behavior in two types of memristor devices, conventional first-order and recently proposed second-order memristor devices, will be first introduced. Secondly, experimental demonstration of *K*-means clustering through unsupervised learning in a memristor network will be presented. The memristor based hardware systems achieved high classification accuracy (93.3%) on the standard IRIS data set, suggesting practical networks can be built with optimized memristor devices. Thirdly, implementation of a partial differential equation (PDE) solver in memristor arrays will be discussed. This work expands the capability of memristor-based computing hardware from ‘soft’ to ‘hard’ computing tasks, which require very high precision and accurate solutions. In general first-order memristors are suitable to perform tasks that are based on vector-

matrix multiplications, ranging from *K*-means clustering to PDE solvers. On the other hand, utilizing internal device dynamics in second-order memristors can allow natural emulation of biological behaviors and enable network functions such as temporal data processing. An effort to explore second-order memristor devices and their network behaviors will be discussed. Finally, we propose ideas to build large-size passive memristor crossbar arrays, including fabrication approaches, guidelines of device structure, and analysis of the parasitic effects in larger arrays.

Chapter 1. Introduction

1.1 Neuromorphic Computing

The concept of neuromorphic computing was first conceived by Carver Mead [1]. It aims to employ electronic circuits based on digital and analog components or often mixed-signal VLSI to mimic the neurobiological structures in nervous systems, and has attracted broad interest as a promising approach for future computing applications [2], [3]. The interest is driven both from the bottom-up, where continued performance gains following Moore's law have become increasingly harder to obtain [4], [5], and from the top-down, where prolific applications of data-centric tasks such as artificial intelligence [6]–[8], big data analysis [9]–[11] and numerical simulations [12], [13] require constant movement of large amounts of data and demand computer architectures that can efficiently address the von-Neumann Bottleneck [14]. Neuromorphic computing allows massive amounts of data to be processed in parallel, potentially with minimal data movement, thus offers a promising solution for current and future computing needs.

To achieve the desired bio-inspired computation, a hardware system would ideally be developed by mimicking morphology and mechanism of biological neural components such as neurons, synapses, and their circuits as well as the overall architectures. Tremendous progress has already been made using conventional analog- and digital integrated circuits to mimic behavioral features of neural elements [15]–[18]. Recent advances in the understanding and development of resistive switching (RS) devices, however, provide an intriguing alternative to directly emulate the

morphology and complex functions of neurobiological components and circuits with high network connectivity, compute density and power efficiency.

1.2 Resistive Switching (RS) Devices

RS devices, often termed memristors or memristive devices, are two-terminal dynamic resistive systems with an inherent memory effect [19]–[21]. Hysteretic RS behavior is not a new physical phenomenon with the first report dating back to the 1960s [22], but interest in RS devices as future electronic components has surged recently due to continuous advances in nano-fabrication, measurement and modeling efforts that greatly improved the RS properties [23]–[30]. A typical device consists of a pair of electrodes (e.g. top and bottom electrodes) sandwiching a switching material, while an array of such devices can form a crossbar structure with a device formed at each crosspoint, as schematically shown in Figure 1-1. Intuitively, the two-terminal

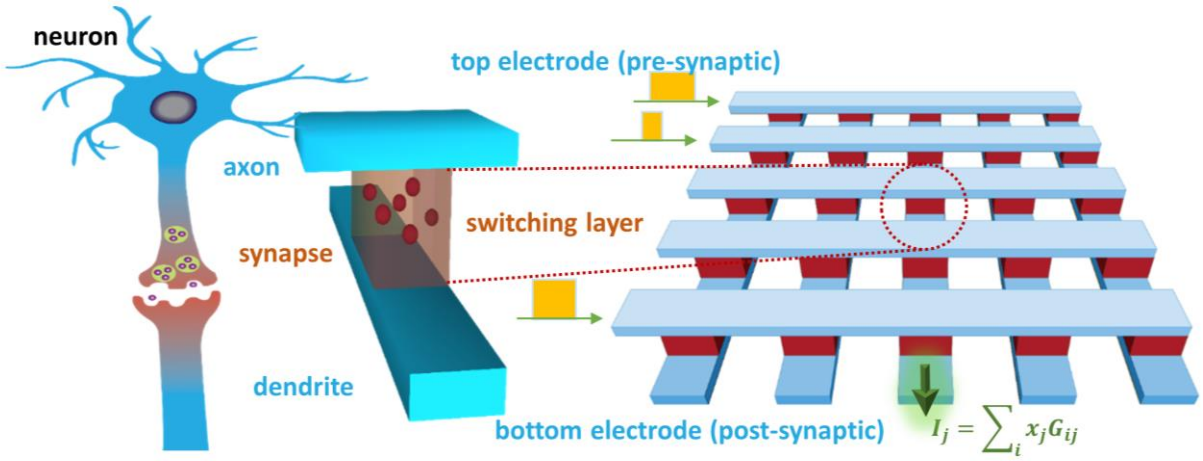

Figure 1-1: Illustration showing the analogy between biological and artificial synapses. The internal ionic dynamics in memristors can be used to faithfully emulate synaptic plasticity effects. A memristor crossbar network can efficiently process input data by performing the weight storage and multiply-and-accumulation functions in the same devices through physical laws.

device resembles a synapse in biological neural systems, where the two electrodes act as an axon and dendrite that connect pre- and post-neurons, with the conductance of the switching layer playing the role of the weight in a synapse [31]–[33]. The crossbar array structure further allows physical mapping of the neural network in hardware and facilitates parallel data processing such as computationally expensive matrix operations directly in memory. As shown in Figure 1-1, when the crossbar network receives input (voltage) pulses having pulse amplitude (or duration) proportional to the values of input vector *x*, the current through a cell at each crosspoint corresponds to the multiplication of the input x_i and the conductance (weight) G_{ij} of that cell, and therefore, the total current (or charge) obtained at column *j* equals to $I_j = \sum_i x_i G_{ij}$, via Kirchhoff's current law and Ohm's law. The output currents from all columns can also be measured simultaneously in parallel, producing the desired output vector. As a result, through a single read operation the crossbar network directly performs the vector-matrix multiplication (VMM), where the weight storage and multiplication-accumulate functions are performed in the same devices. The co-location of the memory function and data processing units and the high parallelism offered by the crossbars are two of the most important features that allow the memristor crossbars to efficiently process data intensive tasks [34]–[38].

RS behaviors in memristive devices as shown in Figure 1-2 are typically caused by redistribution of ions (e.g. oxygen vacancies, V_Os, or metal cations) inside the switching layer. For example, upon the application of a stimulation voltage, V_Os may be accelerated and drift along the field direction. The voltage may also cause a temperature increase that can accelerates V_Os drift and diffusion. After the stimulation, spontaneous ion diffusion may still exist. These processes can occur at different time scales and can lead to rich device dynamics in these seemingly simple two-terminal structures. The V_Os distribution, which determines the local conductivity and the overall

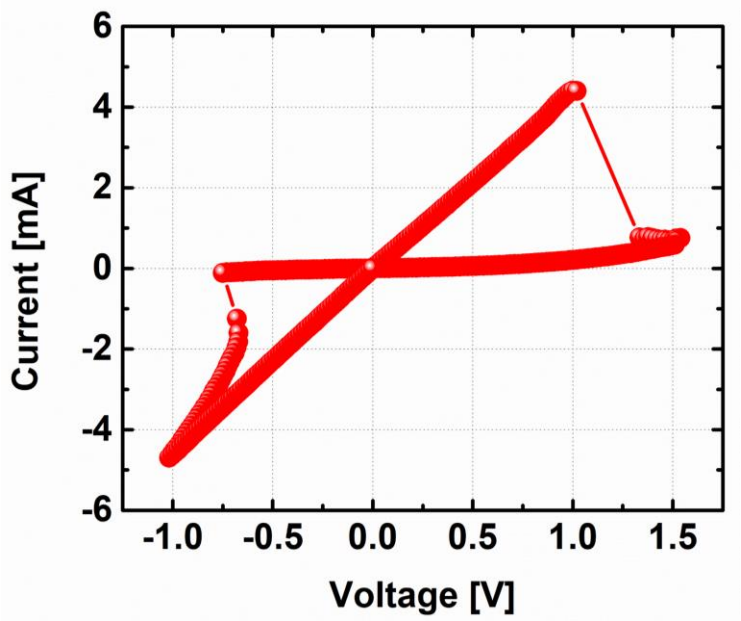

Figure 1-2: A typical DC *I-V* hysteresis curve of a TaOx-based memristor.

device conductance (resistance), is a result of cumulative V_Os drift and diffusion processes and thus depends on the history of the stimulation and results in the apparent memory effects. Similar to synaptic plasticity effects observed in biological systems, the conductance values of memristive devices can be tuned by controlling the stimulation conditions (e.g. voltage pulses and relative timing) from the pre- and post-(artificial) neurons to induce the desired internal ion configurations. Specifically, the internal ion migration dynamics provide opportunities to not only phenomenologically emulate synaptic plasticity effects, but also faithfully mimic short-term and long-term synaptic dynamics in a bio-realistic fashion. In other words, memristor device in artificial neural networks can potentially emulate both the morphology of neurobiological circuits and the underlying molecular and cellular dynamic processes that fundamentally induce functions of the networks [39].

1.3 First-order and Second-order Memristor Systems

RS effects can typically be modeled by a set of coupled equations governing the dynamic ion migration processes(e.g. the ionic continuity equation) and the electronic transport processes (e.g. the electronic continuity equation), respectively [40], [41]. The ionic processes modify the

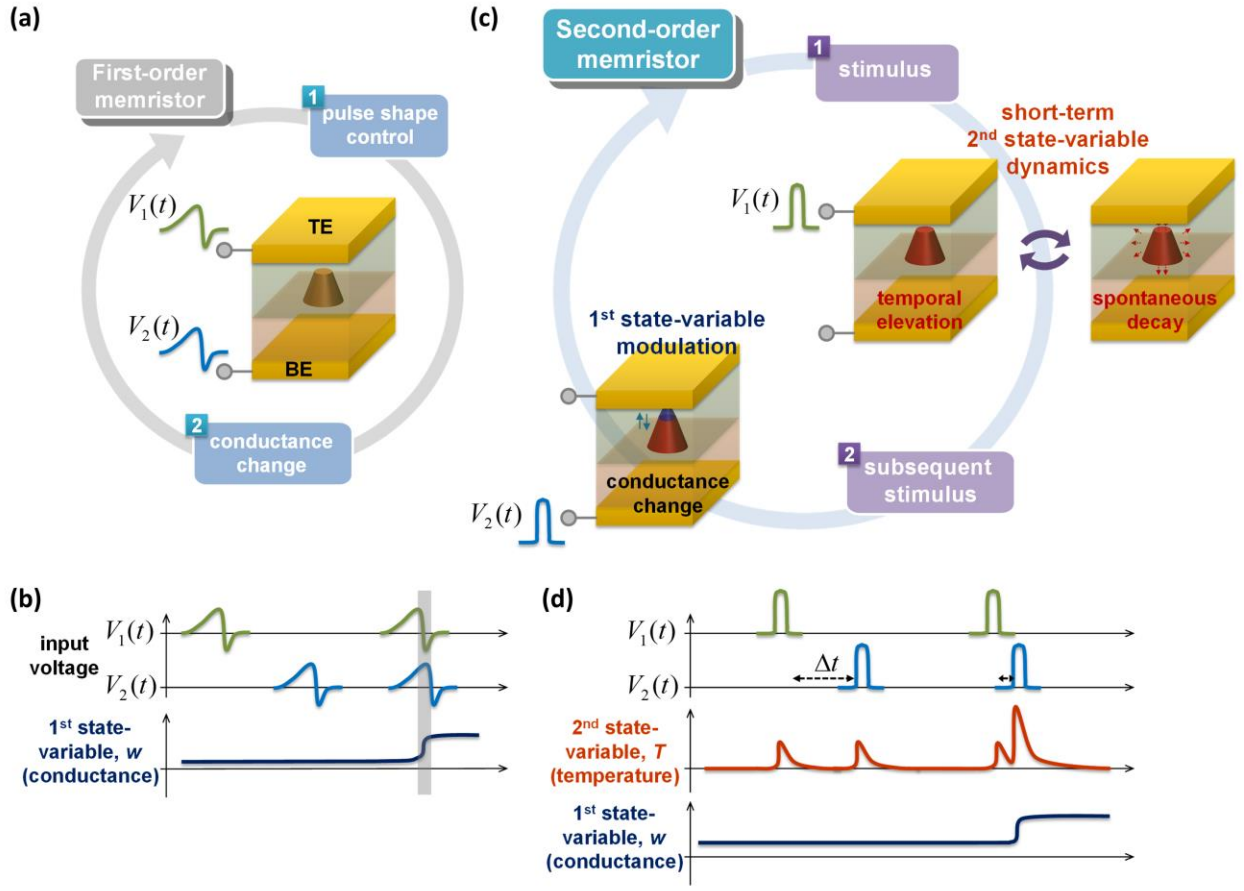

Figure 1-3: Schematics of first-order and second-order device memristor devices. (a) Operation of a first-order device based on a single state variable. (b) The first-order device relies directly on the external stimulus for the conductance change. To encode timing information, carefully engineered pulse shapes are needed to convert different timing into different pulse amplitude or width through overlapping pre- and post-spikes. (c) Operation of a second-order device where the short-term dynamics of one state variable affects the evaluation of the other. (d) By taking advantage of the internal dynamics, the device can directly encode and process different temporal data, using simple, non-overlapping inputs. (Images courtesy of [42].)

local conductivity and can result in conductive channel formation which in turn modifies the electron transport process. To first order, the conductive channel formation process can be modeled by the evolution of an internal state variable, *w* (representing the conducting channel width or length). The dynamics of *w* is a function of the external input (e.g. amplitude and duration of the voltage pulse) and the state of the device (e.g. *w* itself), as described in Eq. (1-1).

The model based on the evolution of one state variable (*w*) can be used to explain the majority of RS devices. We call it a first-order model as there is only one state variable (*w*) and external stimulus directly affects the change of the internal state variable [19], [20], [42], [43], as shown in Eq. (1-1).

$$\frac{dw}{dt} = f(w, V, t) \qquad (1\text{-}1)$$

The current-voltage characteristics of the device can be obtained by self-consistently solving the ionic equation (Eq. (1-1)) with the electron transport equation (Eq. (1-2)).

$$I = G(w, V) \cdot V \qquad (1\text{-}2)$$

The first-order model, schematically illustrated in Figure 1-3(a) [42], has been widely accepted and used to capture the switching characteristics under DC or pulse-based measurements [40], [41].

On the other hand, in Eq. (1-1), the change of state variable *w* is directly determined by the input. The lack of internal dynamics in Eq. (1-1) means that the device cannot be used to directly emulate more complex synaptic behaviors such as spike timing dependent plasticity (STDP) and spike rate dependent plasticity (SRDP) [44]–[46]. To achieve STDP or other timing- or rate-related effects, carefully designed voltage pluses have to be employed to convert the timing and rate information into pulse width or height information through overlapping pulses [31], [33] (Figure 1-3(b)) [42]. This approach works well if the algorithm is well-defined. However, engineering pulses with desired pulse or height is not trivial. Additionally, implementing only a specific

learning rule may also miss other effects that may have not be captured by the phenomenological rules but are critical for the circuits' functions.

Rather than engineering the inputs and following phenomenological "rules", in biology the broad range of synaptic plasticity effects are instead natively regulated by internal dynamic processes, e.g. those involving calcium ions (Ca^{2+}) whose concentration increases temporarily after receiving an action potential then is followed by a spontaneously decay [47], [48]. When a second spike arrives before the Ca^{2+} concentration decays to the resting value, the cumulative effects of the inputs can result in a high enough Ca^{2+} concentration that initiates another molecular/cellular process such as the generation of plasticity-related proteins (PRPs) that results in synaptic potentiation or depression [49], [50]. The biological system can thus be modeled as a system with multiple state variables, with one determining the synaptic weight (e.g. PRP concentration) that is dependent on the other (e.g. Ca^{2+} concentration), with the short term Ca^{2+} dynamics naturally encoding the relative timing information of the spikes and leading to different timing- and rate-dependent plasticity effects.

We call memristor devices having similar multiple state-variables with short-term and long-term dynamic effects second-order memristor devices [42], [51], in contrast to the first-order devices described by Eq. (1-1). Specifically, in a second-order memristor device, the first-order state variable determining the synaptic weight, *w*, is in turn modulated by the inputs along with another, second-order state variable, such as the internal temperature *T* that offers short-term dynamics:

$$\frac{dw}{dt} = f(w, T, V, t) \qquad (1\text{-}3)$$

$$\frac{dT}{dt} = h(w, T, V, t) \qquad (1\text{-}4)$$

, where as usual *w* typically represents the conducting channel size (width or length) and the second-order state variable *T* offers Ca^{2+}-like short-term dynamics and modulates the dynamics of *w*. In the next a few chapters we discuss typical device characteristics and network demonstrations using the conventional first-order devices and second-order devices. While networks based on conventional memristors can effectively process static data, networks based on second-order devices allow more faithful emulation of biological networks and offer potential to efficiently process dynamic, temporal inputs.

1.4 Organization of Thesis

The first chapter introduces different types of RS devices, with an emphasis on the recently-proposed second-order memristor device concept, which offers multiple internal state variables with different time scales that can natively mimic dynamic synaptic processes.

Chapter 2 and 3 discuss network level implementations of neuromorphic systems based on "conventional" (first-order) memristor devices that make use of a crossbar network for acceleration of computation. Chapter 2 demonstrates a common unsupervised machine learning task, *K*-means clustering based on Euclidean distance comparison, while chapter 3 studied a memristor-based partial differential equation (PDE) solver. Both applications are implemented using memristor network hardware.

The next chapters discuss second-order memristor devices. While networks based on conventional memristors already exhibit excellent potential for tasks such as numerical computation, data clustering and classification, second-order memristor devices can naturally encode temporal data using device dynamics as well as provide an opportunity to optimize device characteristics.

In chapter 5 and 6, we discuss challenges for the development of practical memristor array-based hardware systems in terms of fabrication in chapter 5 and network operation in chapter 6, respectively.

Finally, future works including approaches for device optimization and further network applications using memristor crossbar are briefly discussed in Chapter 7.

Chapter 2. *K*-means Data Clustering

Since the vector-vector dot-product (and vector-matrix multiplication) operation constitute the fundamental operation in many machine-learning algorithms, memristor crossbar arrays offer an efficient hardware platform to implement these algorithms. Conductance tuning and vector-matrix multiplication (VMM) can be achieved using "conventional", first-order memristors, where a single state variable is needed to control the device conductance. In the Chapter 2 and 3, we discuss two examples based on (first-order) memristor arrays, *K*-means clustering and Partial Differential Equation (PDE) solvers in a crossbar network that offers massive parallel data processing with minimal data movement.

2.1 Unsupervised Learning and Similarity Indicator

The ability to efficiently implement VMM functions make memristor hardware well suited to map highly developed machine learning algorithms, which have been extensively studied and drove the resurgence of artificial intelligence [36], [37], [52]. Among the approaches, unsupervised learning is of growing importance since it relies on an unlabeled training data set [53], which is far cheaper to obtain than those required by supervised learning algorithms [54], [55]. During training, unsupervised learning rules rely on indicators of similarity between the input feature vectors and the learned feature vectors (dictionary elements), e.g., distance between these vectors in Euclid space, to identify the dictionary element that best matches the input and subsequently adjust the weights accordingly [56]–[58]. If the dictionary elements are normalized, finding the shortest Euclidean distance is equivalent to finding the smallest dot-product between the input

vector and the dictionary element vector, which can be readily obtained in a memristor crossbar [36], [37], [59]. However, when the dictionary elements cannot be normalized during learning, finding the shortest Euclidean distance is no longer trivial, and can cause significant overhead in hardware implementation.

In this chapter, we propose and experimentally demonstrate an approach that directly compares the shortest Euclidean distance in a memristor crossbar hardware system, without normalizing the weights. As a test case, we use this approach to implement the *K*-means clustering algorithm [56], one of the most widely used unsupervised methods in cluster analysis [60], [61], experimentally in a memristor crossbar array.

2.2 Mapping *K*-means clustering onto Memristor Array Network

The *K*-means clustering algorithm aims to partition a set of vector inputs into *K* clusters through exploratory data analysis, as schematically shown in Figure 2-1(a) (for $K = 3$). From the random initial centroid locations, the network evolves as input data points are assigned to different clusters based on the distances between the input data point and the different centroid locations, followed by updating the centroid locations of the clusters. With a relatively simple form, *K*-means clustering provides a comparable solution to more complex approaches such as autoencoders [62] for pre-clustering of unlabeled data set through online training, and reduces the original input space into disjoint smaller subspaces for subsequent use of fine clustering algorithms or data classification through another supervised layer [63], [64].

However, there are two challenges for experimentally implementing algorithms such as *K*-means clustering in memristor-based systems. The first challenge is obtaining the Euclidean distance directly in hardware. Memristor arrays can readily implement VMM operations. However, when the weights are not normalized, the feature vector that produces the largest dot- product with

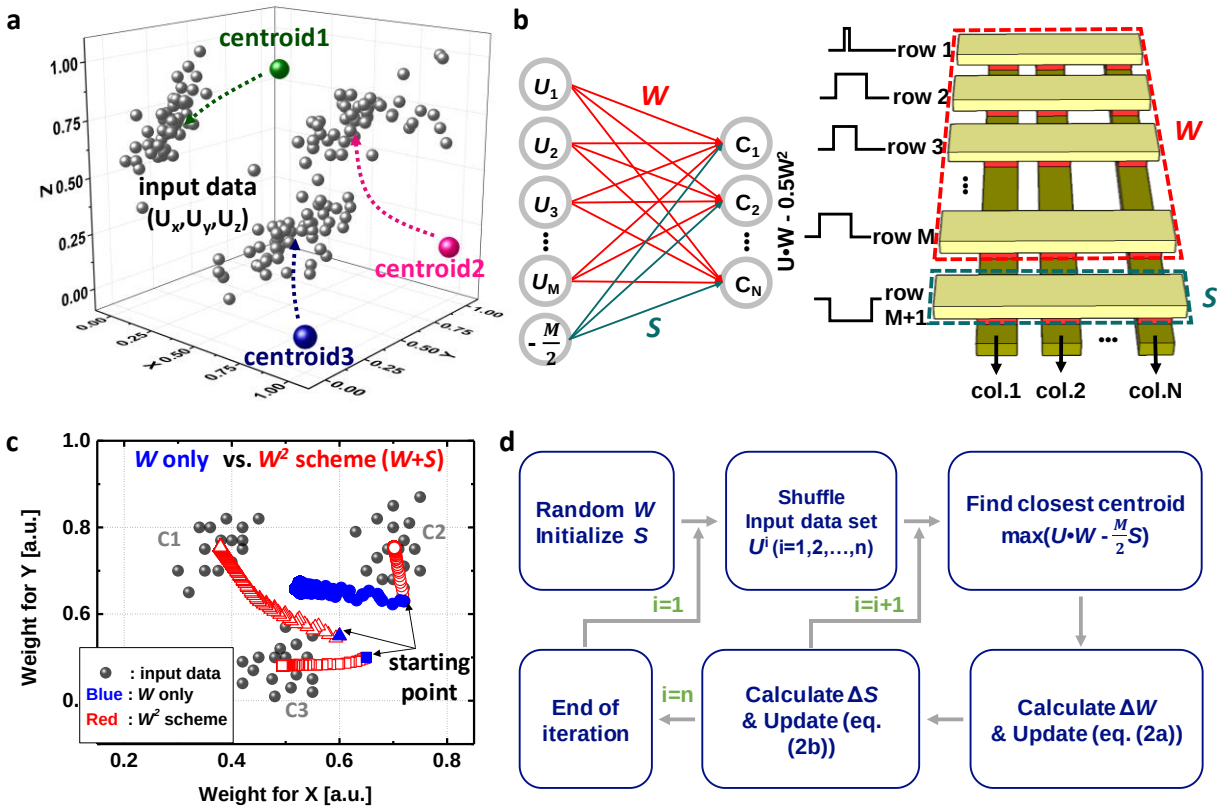

Figure 2-1: Experimental setup of *K*-means clustering implementation. (a) Schematic of the *K*-means clustering algorithm showing the evolution of the *K* centroid locations during online learning. (b) Mapping the proposed algorithm onto the memristor array. The coordinate of a centroid is stored as the memristor conductance values in the corresponding column in the *W* matrix. The W^2 information for the centroid is stored by the memristor conductance in the *S* matrix in the same column. The input values are coded as pulses with different widths and are applied to the rows of the expanded *W* matrix. The accumulated charges at the columns' outputs allow direct comparison of the Euclidean distances. (c) Simulation showing the proposed W^2 scheme can lead to correct centroid evolutions, whereas without the *S* matrix (blue circle symbol) only one centroid was trained. (d) Flow chart of the online learning algorithm. Both the searching and updating operations are implemented in memristor hardware.

the input vector is not necessarily the one having the shortest distance to the input, as will be discussed in detail below. The second challenge is related to the fact that *K*-means clustering is not

aimed at minimizing an error/cost. In previously demonstrated systems, the operation of the memristor network is to minimize a cost function based on the learning rule: either the output label error for supervised learning [36] or the cost function for representing the input in certain unsupervised learning cases [37]. As a result, the algorithm relies on a known reference (either the output label or the original input) to calculate the error/cost. The reference in turn provides a feedback mechanism that helps the network converge near the targeted solution, even in the presence of sizable device variations [65], [66]. This feedback mechanism, however, is missing in many unsupervised learning and arithmetic operation processes, including the *K*-means clustering, since such algorithms do not explicitly aim to minimize an error against a reference, and thus hardware demonstration will post more stringent requirements on the device performance.

Here we address these challenges and experimentally implement the *K*-means clustering algorithm in memristor crossbar array-based hardware as a test case. In the implementation, locations of the *K* centroids are directly mapped as weights in the memristor array in column-wise fashion, i.e., W_{mn} at crosspoint (*m*, *n*) in the memristor array corresponds to the *m*th coordinate value of the *n*th centroid. The weights are in turn represented by the state variable (*w*) in the device model (Eq. S1 and Eq. S2 in the Appendices), which is linearly mapped onto the device conductance. With the input vector set *U* and a randomly generated initial weight matrix *W*, the *K*-means clustering iteratively performs two successive operations to assign the inputs to the appropriate clusters. The first process, the search step, is to find the nearest centroid for a given input and assign the input to the associated cluster. It is then followed by the second process, the update (learning) step, which updates the selected centroid coordinates due to changes in the cluster composition. The crossbar structure as shown in Figure 2-1(b) is extremely efficient in implementing VMM operation, which is one of the core operations in neuromorphic systems,

through simple Ohm's raw and Kirchhoff's law, i.e., the current (or charge) collected at each output neuron ($I_n = \sum_j V_j \cdot W_{jn}$) represents the dot-product of the input voltage vector and the stored conductance matrix. In general, the dot-product operation provides a good indication of the similarity between the input vector and the stored vector, and thus can be viewed as performing a pattern-matching operation and is commonly used in machine-learning algorithms, particularly if the stored feature vectors can be readily normalized. However, algorithms such as *K*-means clustering rely on comparison of the Euclidean distances, not just the similarity between the vectors, to decide the winning neurons and perform the updates.

The Euclidean distance of the input vector U and the weight vector W_n for the nth centroid is determined by Eq. (2-1)

$$\|U - W_n\|^2 = (U^2 - 2U \cdot W_n + {W_n}^2) \qquad (2\text{-}1)$$

Beside the dot-product term $U \cdot W_n$, two other terms, U^2 and W_n^2, are required to obtain the Euclidean distance $\|U - W_n\|$. The U^2 term depends on the input only and thus is a constant for all centroids and will not affect the comparison when determining the winning neuron. However, the $W_n^2 = \sum_j W_{jn}^2$ term, representing the *L2* norm of the weight vector associated with centroid *n*, can in general be different for each centroid. Only in certain conditions will the weight vector be naturally normalized (e.g. when the weight update follows the Oja's rule [66]). Numerically normalizing the weight vectors is expensive, and if not performed, the dot-product $U \cdot W_n$ will not in general correctly represent the Euclidean distance due to differences in the W_n^2 term.

Below we show that Euclidean distance comparisons can still be obtained in memristor crossbar-based implementations. We note that if the array matrix is expanded to include one additional row representing the W^2 term (called the W^2 scheme approach for convenience), results from Eq. (2-1) can still be obtained in the memristor array during a single read operation using the

same VMM approach. Here, the new matrix consists of both the original *W* matrix (sized *M*×*N*) and a new *S* matrix (sized *1*×*N*) (Figure 2-1(b)). The *S* matrix in this case is just a single row and stores the average value of the squared weight ($<W_n^2> = \sum_j(\frac{W_{jn}^2}{M})$) for the n^{th} centroid. The key is then to obtain the desired output using this expanded matrix and to allow the *S* matrix to be updated correctly during unsupervised online learning.

During Euclidean distance comparison, the input vector will now include the original input *U* (applied to the rows of the original *W* matrix), and an additional, constant element -*M*/2 that will be applied to the *S* matrix, as shown in Figure 2-1(b). Here *M* is the number of rows in the *W* matrix. The choice of the input and the *S* matrix is to ensure the values of the elements in the *S* matrix are in the same range as the values in the *W* matrix, so that the same type of memristor devices can be used for the expanded matrix. Experimentally, the elements in the input vector are implemented with voltage pulses having a fixed amplitude (V_{read}=0.3V in our study) and variable widths proportional to the desired input values. The input voltage pulses are applied to the network to perform the vector-matrix multiplication, and the charge Q_n accumulated at each output will represent the distance between the original input vector *U* and the weight vector W_n since $Q_n = \sum_j(U_j \cdot W_{jn} - \frac{W_{jn}^2}{2})$ differs from the exact expression (2-1) only by a (-) sign and a constant U^2. As a result, the higher the output charge obtained from a particular column in the memristor array, the shorter the distance is between the input *U* and the centroid represented by the column.

After identifying the nearest centroid, the second step is to update both the weights associated with the centroid (representing the coordinates of the centroid location) in the original *W* matrix and the *S* matrix representing the $<W^2>$ term. We have developed a learning rule that can be readily implemented in the memristor hardware, shown in Eq. (2-2), and the derivation is represented in the next section.

$$\Delta W_n = \eta \cdot (input - W_n) \qquad \text{(2-2 (a))}$$

$$\Delta S_n = \sum_j (\frac{W_{jn}^2}{M}) - S_n \qquad \text{(2-2 (b))}$$

Here η is a constant and represents the learning rate for the *W* matrix. Note update is only performed for the column corresponding to the nearest centroid, and both the *W* matrix and the *S* matrix are updated simultaneously. By repeating the iterative training process with the searching step and the updating step as described in Figure 2-1(d), the network stabilizes and learns the centroids of the *K* clusters.

To verify the proposed method, we first performed simulations based on a realistic device model (Eq. S1 and S2 in Appendices). As shown in Figure 2-1(c), without the *S* matrix and simply comparing the dot-products between the input vector and the centroid weight vectors, the network could not correctly identify the nearest centroid, and the centroid that happens to have a high initial weight vector length by chance is always picked and updated due to its larger output of the dot-product. Two of the three centroids never got trained as a result. By contrast, with the same starting conditions, the proposed W^2 scheme correctly identifies the nearest centroid and properly updates the centroid locations, leading to successful clustering of the data after 30 iterations.

Note that from a mathematical point of view the added *S* matrix can be considered as a bias effect in neural networks. Implementation of the bias term in memristor networks has been reported previously [36]. However, there are important differences. The *S* matrix has specific physical meaning – W^2, and requires a different training algorithm than the weight update, compared with generic bias terms added to the network output. As such, direct comparison of the Euclidean distances can be obtained in our proposed approach, enabling the experimental implementations of algorithms such as *K*-means clustering in memristor arrays for the first time.

2.3 Weight Update Rules

The *K*-means algorithm aims to group unlabeled datasets into *K* clusters through two steps: step 1. searching the nearest cluster for a given input; and step 2. updating the centroid information for the identified cluster. In our implementation, the centroid locations are stored in the *W* matrix in the memristor crossbar array. Each datapoint to be analyzed is an *M* dimensional vector, e.g., input U_p = (u_{p1}, u_{p2}, … , u_{pM}). For a cluster having q datapoints, the centroid location W^q is calculated as:

$$W^q = (\frac{\sum_{j=1}^{q} u_{j1}}{q}, \frac{\sum_{j=1}^{k} u_{j2}}{q}, \dots, \frac{\sum_{j=1}^{q} u_{jM}}{q}) \qquad (2\text{-}3)$$

Given equation (2-3), when the number of datapoints changes from *q-1* to *q*, the position of the centroid moves by ΔW^q from the prior location by:

$$\Delta W^q = W^q - W^{q-1} = (\frac{\sum_{j=1}^{q} u_{j1}}{q} - \frac{\sum_{j=1}^{q-1} u_{j1}}{q-1}, \frac{\sum_{j=1}^{q} u_{j2}}{q} - \frac{\sum_{j=1}^{q-1} u_{j2}}{q-1}, \dots, \frac{\sum_{j=1}^{q} u_{jM}}{q} - \frac{\sum_{j=1}^{q-1} u_{jM}}{q-1}) \quad (2\text{-}4(a))$$

Examining the first element of ΔW^q , $\Delta W_1^q = \frac{\sum_{j=1}^{q} u_{j1}}{q} - \frac{\sum_{j=1}^{q-1} u_{j1}}{q-1}$ (2-4(b))

$$= \frac{q u_{j1} - \sum_{j=1}^{q} u_{j1}}{q(q-1)}$$

$$= \frac{q u_{q1}}{q(q-1)} - \frac{W_1^{q-1}}{q} - \frac{u_{q1}}{q(q-1)}$$

$$= \frac{u_{q1}}{q} - \frac{W_1^{q-1}}{q} = \frac{1}{q}(u_{q1} - W_1^{q-1}) \sim \eta(u_{q1} - W_1^{q-1})$$

, where if q (number of datapoints in the cluster) is large enough, $1/q$ may be approximated by a learning rate (η) with a small fixed value. In the implementation, we used η = 0.075. Expanding the discussion to other elements in *W* leads to

$$\Delta W = \eta \cdot (input - W) \qquad (2\text{-}5)$$

Additionally, the *S* matrix needs to trace the *W* matrix directly, following equation (2-6).

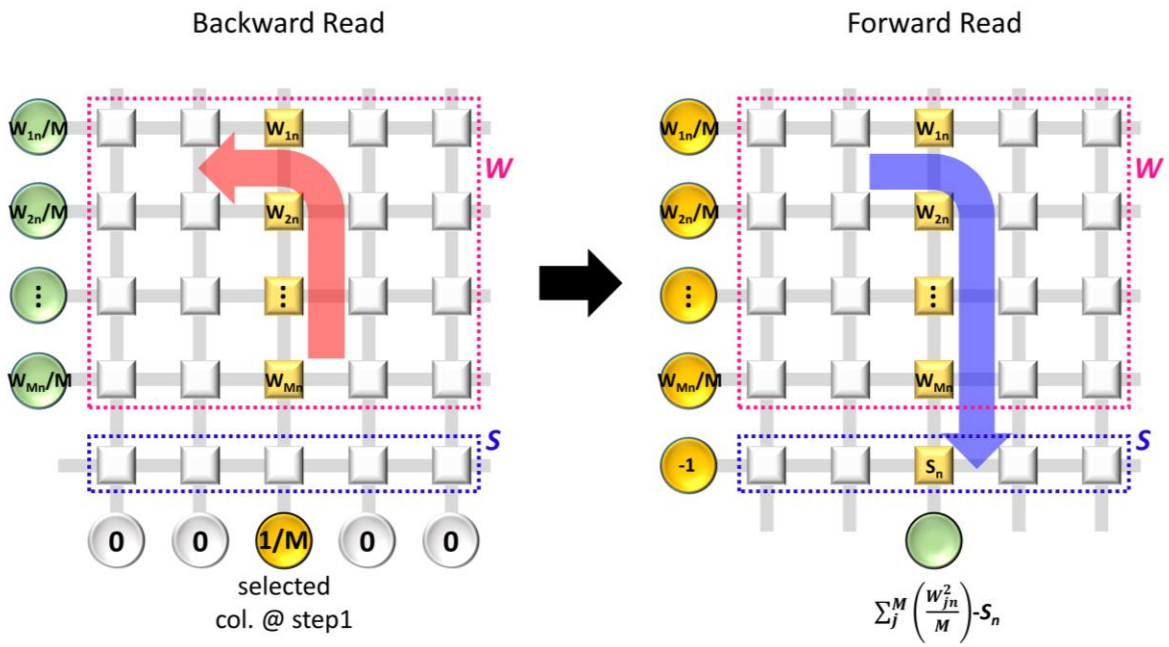

Figure 2-2: Schematics illustrating how ΔS can be obtained from the memristor crossbar.

$$\Delta S_n = \sum_j \left(\frac{W_{jn}^2}{M}\right) - S_n \qquad (2\text{-}6)$$

Experimentally, ΔS_n is obtained from backward and forward read operations in the W matrix to obtain the $\sum_j \left(\frac{W_{jn}^2}{M}\right)$ term, as described in Figure 2-2.

2.4 Analog Memristor Array Implementation

Experimental implementation is based on an optimized Ta_2O_{5-x} based memristor structure having low forming voltage and reliable analog conductance modulations. A scanning electron microscopy (SEM) image of the as-fabricated 16×3 crossbar array is shown in Figure 2-3(a), along with the device structure consisting of a resistive Ta_2O_{5-x} film and a scavenging Ta film (detailed fabrication methods can be found in Appendices). Before reliable switching, the as-fabricated devices generally need to undergo a forming step. However, in a passive crossbar the high voltage used during forming of one device can damage the already-formed, half-selected devices sharing

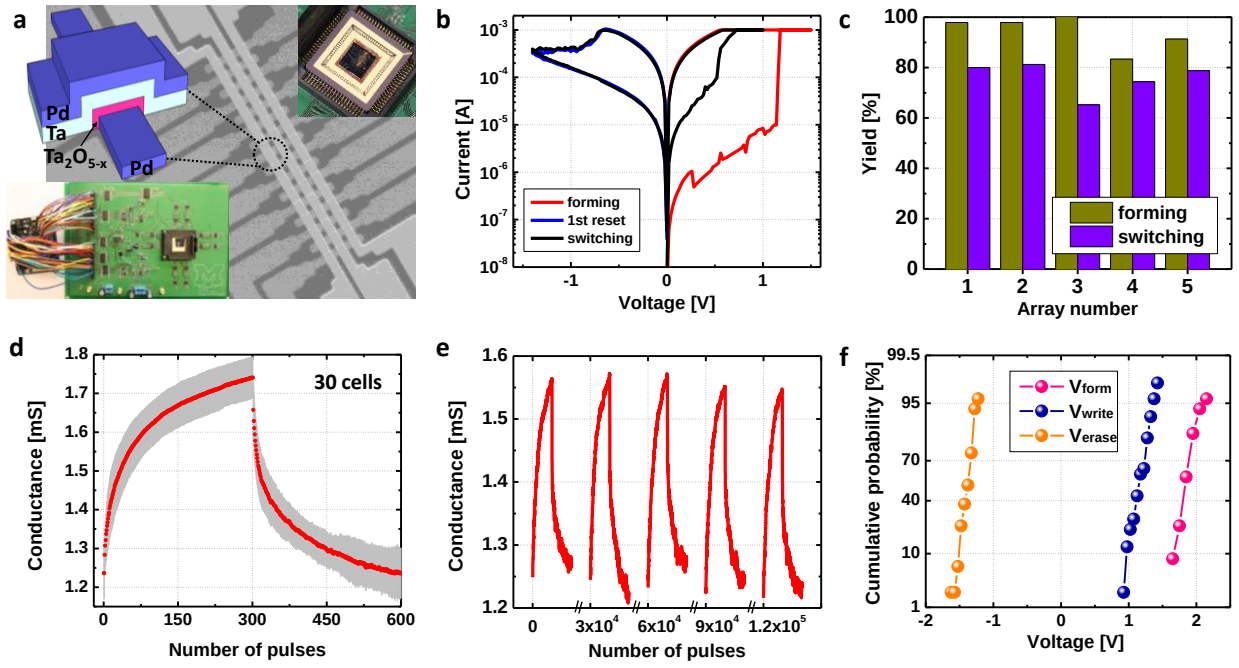

Figure 2-3: Ta_2O_{5-x}-based memristor crossbar characteristics. (a) Scanning electron micrograph (SEM) image of a fabricated crossbar array used in this study. Upper left inset: schematic of the device structure. Lower left inset: Photo of the test board. The memristor array is wire-bonded (upper right inset) and integrated into the board system. (b) DC *I*-*V* curves during the forming cycle and the subsequent switching cycle. The low forming voltage is critical to enable all devices in the crossbar to be properly formed without damaging the devices that have been formed earlier. (c) Device yields from 5 different arrays, showing the percentage of successful forming and the percentage of fully functional devices after all devices in the crossbar have gone through the forming process. (d) Analog conductance update characteristics obtained from 30 devices in the array, showing reliable incremental conductance changes and tight device distribution. Red curve: average conductance during the measurement obtained from the 30 devices. The devices were programmed by 300 consecutive write pulses (1.15±0.1V, 1μs), followed by 300 erase pulses (-1.4±0.1V, 1μs). (e) Cycling curves showing reliable analog conductance updates can be maintained after 1.2×10^5 write/erase pulses. (f) Distribution of the forming, write and erase voltages, obtained from the 16×3 array. We used 4×3 array having narrow voltage range.

the same word-line or bit-line electrode even with a protective scheme. A thin oxide layer can reduce the forming voltage, but can also reduce device yield due to stuck-at-1 (SA1) issues during switching. We addressed this issue by using a thin oxide (3.5nm) deposited with extremely low sputtering power and rate (30W, 1.1Å/min). The low deposition rate allows better control of the oxide film quality and stoichiometry, and helps mitigating the SA1 problem and allows both low forming voltage and high array yield, as shown in Figure 2-3(b)-(c). In addition, a Ta electrode is used to form an oxygen-deficient layer above the Ta_2O_{5-x} switching layer. The Ta electrode was deposited under a low power (100W, 0.5Å/sec) condition to minimize the defect creation in the Ta_2O_{5-x} switching layer (e.g. due to Ta atom injection during deposition). These measures improved the quality of the Ta_2O_{5-x} layer, so that switching will be driven by electrically-controlled oxygen vacancy exchange between the Ta_2O_{5-x} switching layer and the oxygen deficient layer adjacent to the Ta electrode, rather than less controllable intrinsic vacancies in the as-deposited oxide [67], [68].

With these optimizations, improved analog switching behaviors with high yield can be obtained from the memristor array. The long-term potentiation/depression (LTP/LTD) curves of 30 cells from a total of 600 consecutive write and erase pulses are shown in Figure 2-3(d), highlighting gradual and uniform switching behavior. As shown in Figure 2-3(e), reliable analog switching behaviors can be maintained after $1.2x10^5$ programming cycles. Figure 2-3(f) shows the distribution of the forming, write and erase voltage values measured from the 16×3 array, showing tight voltage range (σ=0.1V) for V_{write} and V_{erase}. These devices were used for the *K*-means analysis.

2.5 Experimental implementation of the *K*-means Clustering Algorithm

The *K*-means clustering algorithm was experimentally implemented using the memristor array and a custom-built testing board (Figure 2-3(a)). The test board allows arbitrary pulse signals

to be sent to and electronic current collected from either individual devices or multiple devices in multiple rows and columns simultaneously in parallel.

A simple 2-dimensional (2D) dataset was first used to test the system. Specifically, 50 2D data points that can be explicitly partitioned into three clusters, i.e. K=3, were manually generated in the study to verify the operation of the memristor network. Figure 2-4(a) shows the evolution of the learned centroid positions obtained from the memristor-based K-means system, using online unsupervised training for three different initial weights conditions. During training, the closest centroid to an input is first determined from the memristor array using the W^2 approach, followed by updates of the W and S elements for the selected centroid based on Eq. (2-2). For example, in case (i), the three centroids were initially placed at the same location, *i.e.* having nearly identical W vectors in the memristor matrix (with the small differences due to device variations among the different columns). During training centroid 2 moved in the left direction towards one group, while centroid 3 moved in the up direction. Centroid 1 moved in the upper left direction with relatively large movements at the beginning of training, and turned left towards the group of data farthest from the initial point. After training, the three centroids settled to the centers of the respective clusters, and every point in the dataset can be properly assigned to one of the three clusters. Other initial centroid locations, such as having all centroids located in the center of the dataset (case ii), and having the centroids randomly distributed (case iii) have also been tested, and the memristor network can successfully perform K-means clustering in all cases (Figure 2-4(a)) using our experimental setup, demonstrating the reliability of the proposed training algorithm and the hardware implementation using physical memristor crossbar arrays, even with non-normalized weights.

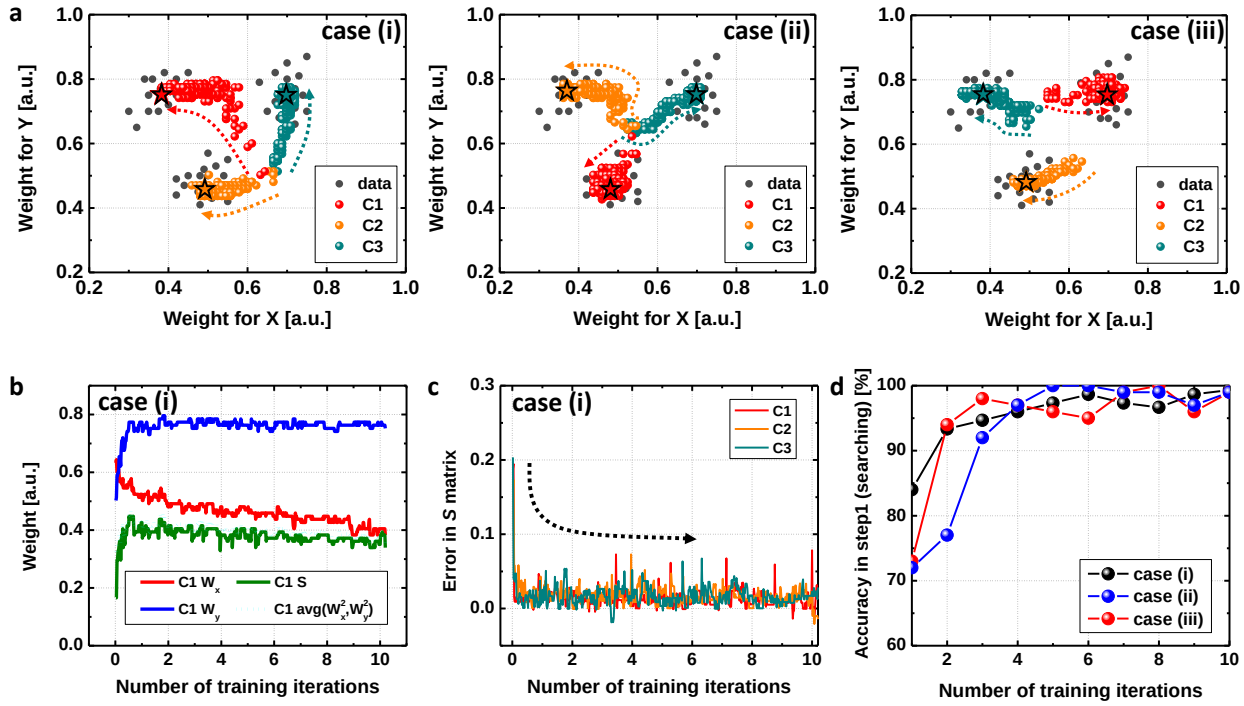

Figure 2-4: Experimental implementation of *K*-means clustering using the memristor crossbar. (a) Evolution of the three centroid locations during training, for three cases with different initial configurations: i) all three centroids were located at the same position outside the dataset; ii) all three centroids were located at the same position inside the dataset; iii) randomly assigned initial positions. The final locations of the centroids are represented by the stars. (b) Evolution of the *W* elements W_x, and W_y (representing the centroid coordinates) for centroid 1 in case I, along with the centroid's *S* element, and a reference showing the calculated $<W^2>$ during training. Each iteration includes 50 training operations for a given input data set. (c) Difference between the stored *S* element and the calculated $<W^2>$ for the three centroids, showing a rapid decrease after only a few training steps. (d) Success rate of correctly finding the nearest centroid using the memristor network, as a function of the iteration number during training. The success rate becomes > 95% after 4 iterations.

We note that the reliability of the *K*-means clustering algorithm implementation depends critically on how accurately the *W* and *S* elements are modulated during training, since errors in these elements directly affect the obtained Euclidean distance and the subsequent identification of

the winning centroid. The experimentally obtained evolution of the W and S elements during training are shown in Figure 2-4(b). One can see that as a W vector (e.g., W_x and W_y of centroid 1 in Case (i) shown in Figure 2-4(b)) is updated when a new data point is added to the cluster, the corresponding S element is properly adjusted based on the proposed algorithm, and the learned S elements in the memristor network following the proposed Eq. (2-2(b)) are indeed in good agreement with the calculated value of $<W^2>$. Specifically, Figure 2-4(c) show the difference between the stored S values and the calculated $<W^2>$ value (numerically calculated from the measured values of the W matrix) in Case (i). The error is large in the beginning because the S elements were not initialized according to the W matrix values. Importantly, the error drops rapidly with only a few training steps and remains very low, due to the excellent incremental conductance modulation capability of the physical device as shown in Figure 2-3(d). As a result, the system can correctly find the nearest centroid with over 95% success rate after 4 iterations as shown in Figure 2-4(d), verifying that the proposed W^2 scheme can be used to efficiently calculate distances between vectors without compute-intensive normalization processes.

2.6 Analysis of the IRIS dataset

Based on the successful analysis of the test dataset, we applied the memristor-based hardware system to perform K-means clustering analysis of the IRIS flower dataset [69], a widely used dataset in machine learning. The IRIS dataset includes data from four features such as the length and the width of the sepal and the petal (inset of Figure 2-5(a)), measured from 150 samples from each of the three iris flower species; *setosa*, *virginica*, and *versicolor*. The goal is to successfully separate the three species based on these measured data. Figure 2-5(a) shows the evolution of the three centroids (representing the three species) obtained from the memristor system during training. Following standard practice, only three features (the sepal width, the petal

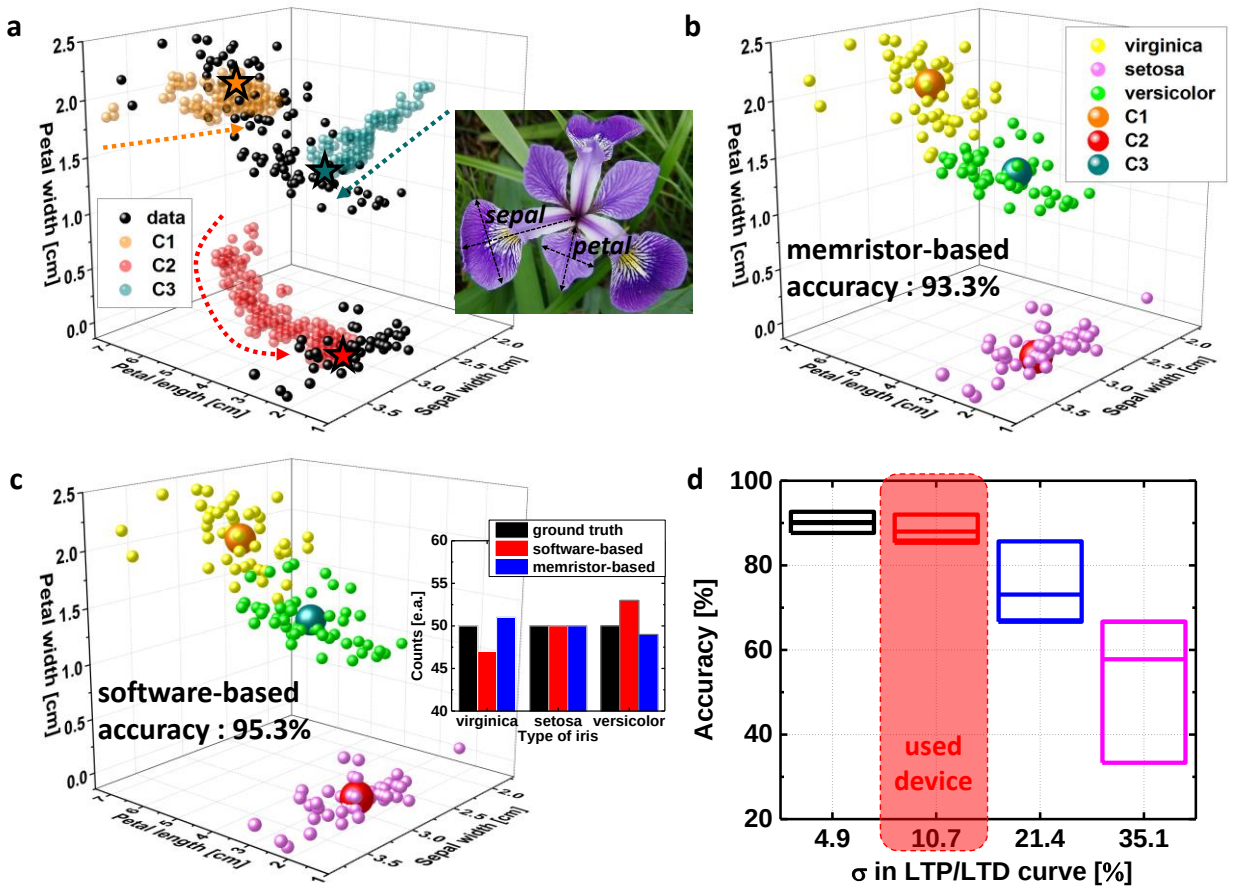

Figure 2-5: *K*-means clustering analysis of the IRIS dataset. (a) Evolution of the centroid locations during online training with the unlabeled 3D IRIS dataset. Inset: features in the IRIS dataset, including the length and the width of the sepal and the petal. In this work, only the three features (the sepal width, the petal width, and the petal length) that produce the highest accuracy were used. (b) Clustering results using the memristor-based network after training. The final positions of the three centroids are represented by large circles, and the datapoints that are partitioned into the three clusters depending on the nearest means are represented by small circles with different colors. (c) Results from *K*-means clustering analysis obtained from software. Inset: Comparison of results obtained from software- and memristor-based methods for the 3 types of flowers, with reference to the ground truth. (d) Simulation results showing the effects on *K*-means clustering analysis accuracy as a function of device variation. High device uniformity achieved in this study is critical to obtain the desired accuracy.

width, and the petal length) that produce the highest classification accuracy were used in the analysis. Even though the boundary between *virginica* and *versicolor* is inherently complex in the IRIS dataset, the three centroids were updated properly in the memristor-system with initially randomized *W* and *S* matrices, and led to a final configuration that enabled proper clustering of the unlabeled data. The final cluster analysis obtained from the trained memristor-based network is shown in Figure 2-5(b), corresponding to a classification accuracy of 93.3%. This experimental result is comparable to the result (95.3%) obtained from a software-based method (Figure 2-5(c)), demonstrating the feasibility of the experimental memristor network system with the proposed W^2 scheme for data-intensive cluster analysis.

Since *K*-means clustering analysis is based solely on the Euclidean distances between the input and the dictionary vectors, and does not rely on minimizing an output label error or a cost function which can provide a feedback mechanism to help network stabilization, more accurate vector-matrix multiplication and weight modulation operations are required. In this case, effects such as cycle-to-cycle and device-to-device variations can significantly affect the accuracy of the clustering analysis during experimental implementation. Importantly, the memristor devices used in this work with improved switching uniformity has a device variation of ~ 10% (characterized by the standard deviation during LTP/LTD measurements), as shown in Figure 2-3(d), which enabled us to achieve the high clustering accuracy observed experimentally. To verify the effect of device variations, detailed simulations that incorporate device variation effects (Figure 2-6) were performed. The accuracy of *K*-means clustering analysis was found to significantly decrease when device-to-device variation is larger than 20% (Figure 2-5(d)), confirming our hypothesis and experimental findings. The high clustering accuracy obtained experimentally demonstrates the potential of memristor-based networks for not only “soft” neuromorphic applications but also for

parameter	nominal value
α	2.12e-3
β	6.01e-1
δ	2.64e-1
γ	6.53e-3
κ	8.91e-4
$\mu 1$	1.79e+1
$\mu 2$	1.42e+1

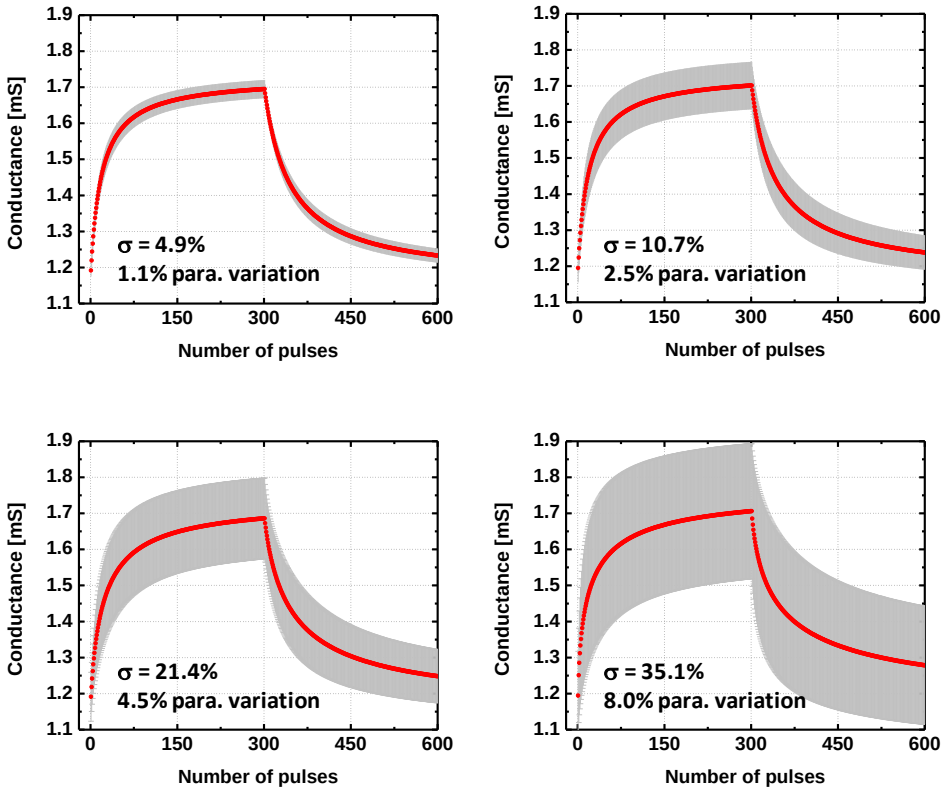

Figure 2-6: Left table: device parameters used in the model. Right: simulated LTP/LTD curves with different device parameter variations. The 2.5% variation case (leading to σ=10.7% in conductance updates during LTP/LTD) quantitatively reproduces the experimental data (σ=10.8% in conductance updates)

arithmetic applications that require high accuracy. By extending the simple dot-product operations between the input and the weights using bias-like terms as discussed in this study, more complex and elaborate algorithms may also be implemented.

2.8 Conclusion

We experimentally demonstrated that memristor-based neural networks can successfully perform *K*-means clustering. A W^2 scheme was proposed to allow accurate calculation of the Euclidean distance, which is an essential operation in many machine learning algorithms, through

direct VMM in memristor networks with non-normalized weights. Without a cost function that provides a feedback mechanism, the *K*-means clustering algorithm poses stricter requirements on compute accuracy and device variability compared with other neuromorphic computing algorithms. With an expanded weight matrix, properly designed training rules, and improved device properties, we show *K*-means clustering can be reliably implemented using memristor networks. The standard IRIS data set was successfully processed through unsupervised, online learning with high accuracy (93.3%). With continued device and algorithm optimizations, the results obtained here can pave the way toward practical memristor network hardware for more broad applications beyond neuromorphic systems.

Chapter 3. Partial Differential Equation (PDE) Solver

In the previous chapter, we showed that *K*-means clustering algorithm is implemented in memristor networks using a proposed extended network structure that allows accurate comparison of the Euclidean distances. In this chapter, we discuss another example that highlight the co-located memory/logic and parallel processing properties of memristor arrays. Specifically, we show that memristor arrays can not only be used in "soft" tasks such as machine learning where approximate solutions are often sufficient, but also in "hard" tasks such as solving Partial Differential Equations (PDEs), where high computing accuracy is necessary.

Numerical computations based on solving PDEs are ubiquitous in scientific research and engineering [70]–[73], and many other tasks that involve simulation, prediction and optimization such as weather forecasting [13] and economics [12]. A PDE is any equation with a function of multiple variables and their partial derivatives. Let u be a function with independent variables $t, x_1, \dots, x_n$, defined as:

$$u = u(t, x_1, x_2, \dots, x_n) \qquad (3\text{-}1)$$

A general PDE of u has the form of:

$$f\left(t, x_1, \dots, x_n, u, \frac{\partial u}{\partial t}, \frac{\partial u}{\partial x_1}, \dots, \frac{\partial u}{\partial x_n}, \frac{\partial^2 u}{\partial t^2}, \dots, \frac{\partial^2 u}{\partial x_n^2}, \dots\right) = 0 \qquad (3\text{-}2)$$

Analytical PDE solutions are rare and the vast majority of systems of PDEs are solved (or integrated) using numerical methods that are computationally expensive, involving iterative VMM operations while handling massive amounts of data.

So far, the focus using memristor-based computing has been on tasks such as artificial neural networks [36], [37], which typically aim to obtain an approximate or qualitative solution and where limited precision and device variabilities can be tolerated [74]–[76]. This is not the case for numerical computational tasks such as solving PDE problems, where high precision and accurate solutions are mandatory, making it more challenging to implement these computing tasks in memristor-based hardware. For example, a well-designed memristor device may provide around 64 different resistance levels [77], [78], which is equivalent to 6 binary bits. However, practical numerical tasks may require up to 64 bits (2^{64} levels) of precision. Additionally, solving PDEs normally involves working with very large matrices that are neither practical nor efficient to fit in a single memristor crossbar.

In this chapter, a memristor-based in-memory computing system to solve PDE is presented. We focus our implementation of a complete hardware and software package that can effectively address the limited device precision and crossbar size concerns.

3.1 High Precision Memristor Computing System

Typically, a system of PDE is solved numerically by discretizing space (and/or time) into grid points such that the partial derivatives at one point can be reduced into combinations of the variable values at several neighboring grid points. Afterwards, the problem is mapped to matrix form, with the numerical coefficients representing linearized operators between variables at neighboring grid points. The resulting coefficient matrix can be very large but is typically sparse. This process is performed during the initial problem formulation stage, using techniques such as finite-difference, finite-element or finite-volume methods. Iterative methods are then used to estimate the variable values at the grid points through the coefficient matrix and the system's

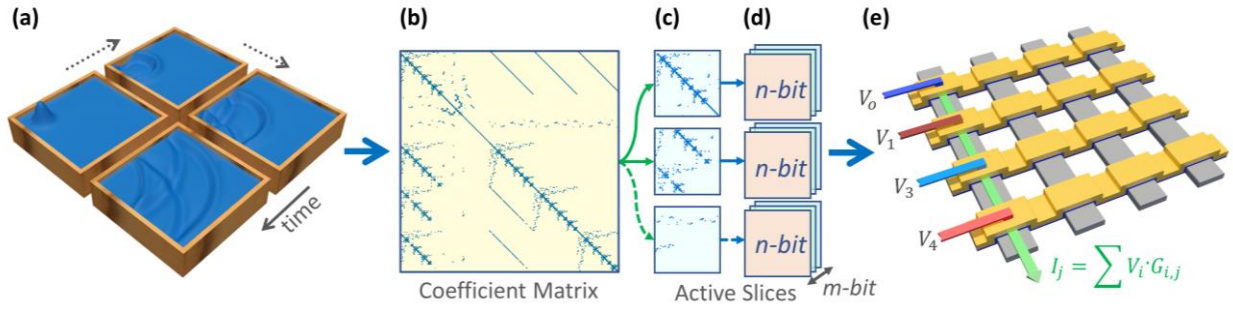

Figure 3-1: High-precision PDE solver based on memristor crossbar. (a) An example of a time-evolving PDE system showing a water wave inside a pool at four different time instances. (b) A sparse coefficient matrix used in numerically solving PDEs. (c) The coefficient matrix is sliced into equal sized patches, and numerical calculations are only performed for the active (nonzero) slices. (d) Each high-precision (m-bit) active slice is mapped into multiple arrays, each representing a portion (n-bit) of the desired precision. (e) The values of the elements in the n-bit slice are mapped as conductance values in a memristor crossbar of the same size, and VMM of the slice is performed by supplying the input vector as voltage pulses to the rows and read out the current outputs at the columns.

boundary conditions (Figure 3-1(a)). These operations can be performed through a series of VMM operations that we aim to compute in memristor crossbars.

For practical systems, the coefficient matrix can be very large, e.g. a 2D system with a 100×100 grid will result in a coefficient matrix with $(10^4)^2=10^8$ elements. However, the coefficient matrix is also typically very sparse, with only a very small fraction of non-zero elements, as shown in Figure 3-1(b). This makes it difficult and inefficient to map the coefficient matrix into a single memristor array. By taking advantage of the sparsity, we can divide the matrix into equally sized slices and map only the active slices (the ones containing nonzero elements) into memristor crossbars, as shown in Figure 3-1(c). By doing so, crossbars with practical sizes, e.g. 16×16 or 32×32 can be used to map the active slices, while significantly improving the hardware utilization.

We also show that the low native precision of memristor devices can be extended through the use of multiple crossbars, where each crossbar represents a given number of bits (Figure 3-1(d)). This precision expansion approach is similar to the techniques utilized in digital circuits, where binary (2-level) physical values, such as capacitor voltages in a dynamic random-access memory, are used as the basis of high-precision computing systems. Precision extension technic will be discussed in the next section in detail.

At the single crossbar level, analog VMM are performed directly between an input vector, represented by the voltage pulses applied to the rows, and the coefficient matrix elements, represented by the memristor conductance values, as shown in Figure 3-1(e). By summing results from the partial products from these base-l operations, the desired output from the extended precision can then be obtained.

3.1.1 Precision Extension Technique

Precision extension is a system level technique proposed to improve the effective precision of memristor-based hardware. In this case, a high precision can be provided by multiple devices together, each of which stores a portion of the required bit-width. The same technique is also used to implement the required precision of the input and output analog data to and from the crossbar array, by representing a high-precision data using multiple splits.

The approach here is to treat the data in a base-l number system, where l is the number of levels represented by a single digit. For example, operations of the 12-bit numbers can be processed in a physical system based on 6-bit devices, e.g. 12-bit vectors X and Y can be represented as:

$$X = \begin{bmatrix} (a_1, a_0)_l \\ (b_1, b_0)_l \\ (c_1, c_0)_l \end{bmatrix} = (X_1, X_0)_l \quad (3\text{-}3)$$

$$Y = \begin{bmatrix} (d_1, d_0)_l \\ (e_1, e_0)_l \\ (f_1, f_0)_l \end{bmatrix} = (Y_1, Y_0)_l \quad (3\text{-}4)$$

where X_1, X_0, Y_1, Y_0 are 6-bit vectors. A dot product between the two vectors is performed as:

$$X \cdot Y = (X_0 \cdot Y_0) + l(X_0 \cdot Y_1) + l(X_1 \cdot Y_0) + l^2(X_1 \cdot Y_1) \quad (3\text{-}5)$$

where l and l^2 donate single and double digit shifts. Each partial dot-product is computed as at the (native) single digit level:

$$X_i \cdot Y_j = a_i d_j + b_i e_j + c_i f_j \quad (3\text{-}6)$$

Equation (3-6) can be directly executed in a memristor crossbar by encoding $\{a_i, b_i, c_i\}$ as the input voltages applied to different rows and $\{d_j, e_j, f_j\}$ as the memristors' conductance values along a column. The results of the partial products are summed together according to Eq. (3-5) to obtain the full dot product result. Other arithmetic operations with the extended precision can be performed similarly by splitting the high-precision operations into partial operations.

3.1.2 ADC Quantization

To properly implement the precision extension technique, the partial products need to be quantized before the digit shift operations, otherwise noise (error) in the analog output due to device variation becomes amplified during the digit shift and reduces the precision of the final output. Fortunately, the quantization operation can be readily implemented in the existing circuitry through the ADC circuitry that is already used in the hardware to quantize analog outputs to digital values. In a typical ADC operation, any analog value within quantization thresholds is represented

by the same quantized value, thus preventing the noise in the analog signal from being amplified during the shift operation, and thus enabling the shift operations to perform properly.

Specifically, the required number of ADC bits are determined by the sum of the input bits, the stored coefficient (weight) bits, and the number of non-zero inputs per column:

$$b_{ADC} = b_i + b_d + \log_2 \omega \qquad (3\text{-}6)$$

where b_i is the number of bits of the input, b_d is the equivalent number of bits of each memristor device, and ω is the number of non-zero inputs per column. By designing ADCs with the right number of bits based on Eq. (3-6) (instead of pursuing as many bits as possible), the ADC area and power consumption overhead can be minimized while also providing the desirable quantization effect to minimize error propagation during the shift operations.

3.1.3 Optimized Device and Write Verify Technique

We experimentally implemented the proposed approach in a hardware based on the optimized Ta_2O_{5-x} memristor crossbar array, which was introduced in *K*-means clustering case. The devices showed narrow distributions (σ<0.1 V) for forming, set, and reset voltages and good analog conductance changes, making these devices well suited for passive crossbar array operations.

However, the inherent stochastic ion migration processes in the set and reset stages lead to sizable device variability as shown in Figure 3-2(c). Without any feedback mechanism during programming, a cell-to-cell variation of 5.3% may occur, limiting the native precision of a single device to around 4 bits. Lower device variability can be obtained by using a write-verify feedback method as shown in Figure 3-2, leading to a cell-to-cell variation of <1%. Specifically, each write

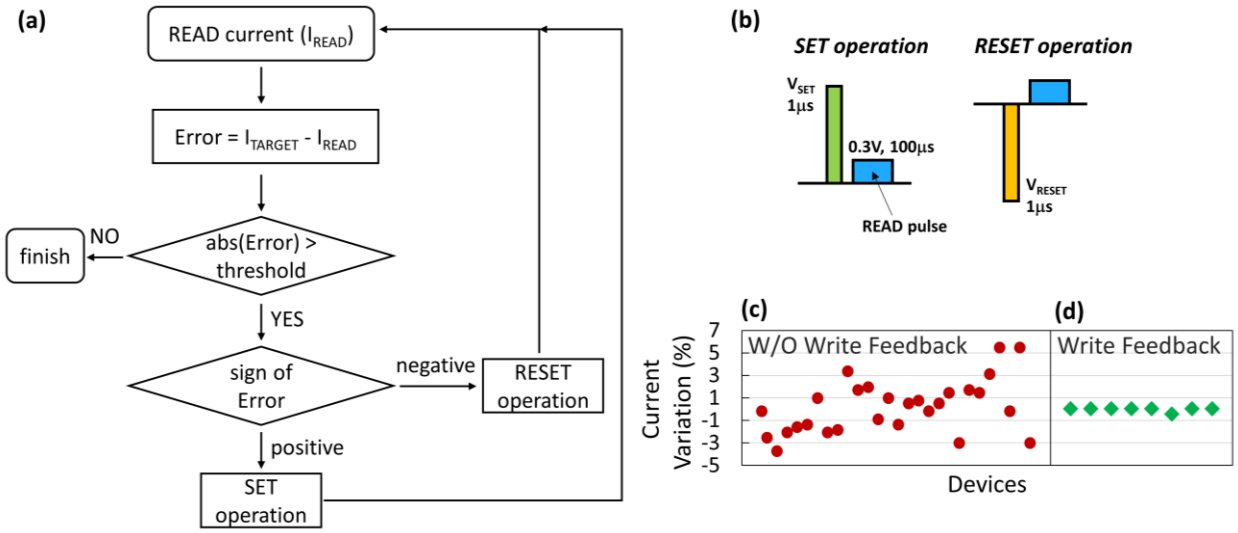

Figure 3-2: Flow chart and results of the write-verify scheme. (a) Flow chart. Current from the Read operation on a target cell is used to compare with a target value and calculate an error (error = current difference). If the current error is below a pre-defined threshold, the operation is considered complete and the process stopped, otherwise operations are taken based on the sign of the error. For positive errors, a SET pulse is applied to increase the device conductance, while for negative errors a RESET pulse is applied to decrease the device conductance. The procedure is then repeated. In the implementation, we used 1% error as the criterion to stop the process, and achieved less than 0.5% errors with 100 write-verify cycles on average when writing to a device initially at HRS. (b) Pulse shape of set and reset operation. (c) Variation of device conductance obtained from pre-determined programming conditions without any feedback mechanism. A 5.3% conductance variation is achieved. (d) With the write-verify approach, the conductance variation is reduced to 0.85%, making it possible to implement the PDE solver in memristor-based hardware.

operation is based on a sequence of write-read pulse pairs, each pair including a programming (set or reset) pulse and a subsequent read pulse (0.3 V, 100 µs) for verification purpose. By comparing the read current with a target value of the cell, the programming pulse in the next pair in the sequence is determined, i.e. set (reset) for conductance increase (decrease). Each set (reset) pulse has a fixed duration (1 µs), but gradually increasing voltage levels to achieve the desired

conductance value (0.75V to 0.9V for set, -0.85V to -1.05V for reset). When the conductance reaches within a pre-determined range of the target value (e.g. 1%), the write operation is considered complete. In the experimental implementation, the initial write sequence (with the device at the HRS state) typically requires 100 write-verify pairs on average, while updating the coefficients later on typically requires around 10 write-verify pairs in a write sequence. Combined with the precision-extension and ADC quantization technique discussed above, this device system is successfully used to experimentally demonstrate the proposed high-precision PDE solver system.

3.2 Poisson's Equation Example

The first test problem using the memristor-based system was static PDE solution. This class of equations describe spatial relationships between the variables at a steady-state condition. Examples include elliptic PDE systems such as Laplace's ($u_{xx} + u_{yy} = 0$) and Poisson's ($u_{xx} + u_{yy} = f(x, y)$) equations [79]. Typically, elliptic and other systems of PDEs can be numerically formulated as solving an $A \cdot X = B$ problem, where X is the unknown vector to be solved, A is the coefficient matrix, and B is a constant vector containing the boundary conditions. While such problems can be solved using several numerical techniques, here we adopted the Jacobi method [80] since it can be directly mapped to the memristor crossbar hardware system using entirely iterative VMM. Based on this approach, we experimentally solved a Poisson's equation test case at 16-bit precision using our hardware setup. The problem is defined as:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2 \cdot \sin(x) \cdot \cos(y) \qquad (3\text{-}7)$$

3.2.1 Poisson's Equation with Finite Difference

The Poisson's equation described in Eq. (3-7) is mapped to the crossbar hardware system by first approximating it using central finite different method (FDM) and a common 2D 5-point stencil as

$$\frac{u_{i+1,j}+u_{i,j+1}-4u_{i,j}+u_{i-1,j}+u_{i,j-1}}{h^2} = -2 \cdot \sin(x_i) \cdot \cos(y_i) \qquad (3\text{-}8)$$

where i is the x-axis grid index, j is the y-axis grid index, h is the distance between two neighboring grid points along the x-axis or y-axis. In an example where the problem's domain is discretized into a 3×3 grid (excluding boundary points), Eq. (3-8) is transformed into a matrix form as:

$$\begin{bmatrix} \mathbf{-4} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{-4} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{-4} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & \mathbf{-4} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{-4} & \mathbf{1} & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{-4} & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{-4} & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{-4} & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{-4} \end{bmatrix} \cdot \begin{bmatrix} u_{0,0} \\ u_{0,1} \\ u_{0,2} \\ u_{1,0} \\ u_{1,1} \\ u_{1,2} \\ u_{2,0} \\ u_{2,1} \\ u_{2,2} \end{bmatrix} = \begin{bmatrix} h^2 f(x_0, y_0) - b_{-1,0} - b_{0,-1} \\ h^2 f(x_0, y_1) - b_{-1,1} \\ h^2 f(x_0, y_2) - b_{-1,2} - b_{0,3} \\ h^2 f(x_1, y_0) - b_{1,-1} \\ h^2 f(x_1, y_1) \\ h^2 f(x_1, y_2) - b_{1,3} \\ h^2 f(x_2, y_0) - b_{2,-1} - b_{3,0} \\ h^2 f(x_2, y_1) - b_{3,1} \\ h^2 f(x_2, y_2) - b_{2,3} - b_{3,2} \end{bmatrix} \qquad (3\text{-}9)$$

where $f(x_i, y_j) = \sin(x_i) \cdot \cos(y_i)$, and $b_{i,j}$ are the boundary values of the system.

Similar matrices can be obtained for larger grids. It should be noted that the sparsity of the coefficient matrix is a function of the mesh size with larger meshes leading to more sparse matrices, since the matrix dimension is determined by the total number of grid points, while the number of nonzero elements equals to the fixed (small) stencil size. A typical stencil size is 5 for second-order 2D PDEs, while the matrix dimension can be very large (e.g. with 900 elements in a row for a small 30×30 grid), leading to very sparse matrices.

3.2.2 Jacobi Method and Mapping

Elliptic PDE system in Eq. (3-9) is in the form of $A \cdot X = B$, and can be solved using the Jacobi method. The Jacobi method is a numerical technique used to solve diagonally dominant linear systems, i.e. $A \cdot X = B$, where:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}, B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \text{and } X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad (3\text{-}10)$$

The iterative solution is obtained from:

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}}{a_{ii}}, i = 1,2, \dots, n \quad (3\text{-}11)$$

Eq. (3-10) can be written in a matrix form as:

$$X^{(k+1)} = D^{-1}\left(B - R \cdot X^{(k)}\right) \quad (3\text{-}12)$$

where D is a matrix with only the diagonal elements of A, and R contains the remaining elements of A. In the case where the diagonal elements are equal, which is common, Eq. (3-12) is simplified as:

$$X^{(k+1)} = \frac{1}{d}\left(B - R \cdot X^{(k)}\right) \quad (3\text{-}13)$$

, where d is the common diagonal element.

We note any scientific or engineering system that includes transport phenomena or reaction chemistry (e.g. fluid dynamics, radiation transport) that is modeled for chemical engineering, combustion, fluid mechanics, solid state physics and nuclear engineering, in addition to plasma physics, is of a diagonally-dominant matrix. Thus, the vast majority of the PDE problems are diagonally-dominant and can be solved by the proposed method in this work. It should be noted, however, that diagonally-dominant coefficient matrices do not have to be structured. An

unstructured matrix can be divided into slices that are diagonally dominant and mapped into the crossbar system as well.

Using the Jacobi method, Eq. (3-9) can be represented as:

$$\begin{bmatrix} u_{0,0} \\ u_{0,1} \\ u_{0,2} \\ u_{1,0} \\ u_{1,1} \\ u_{1,2} \\ u_{2,1} \\ u_{2,2} \\ u_{1,2} \end{bmatrix}^{(k+1)} = C + \begin{bmatrix} 0 & \mathbf{1/4} & 0 & \mathbf{1/4} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1/4} & 0 & \mathbf{1/4} & 0 & \mathbf{1/4} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1/4} & 0 & 0 & 0 & \mathbf{1/4} & 0 & 0 & 0 \\ \mathbf{1/4} & 0 & 0 & 0 & \mathbf{1/4} & 0 & \mathbf{1/4} & 0 & 0 \\ 0 & \mathbf{1/4} & 0 & \mathbf{1/4} & 0 & \mathbf{1/4} & 0 & \mathbf{1/4} & 0 \\ 0 & 0 & \mathbf{1/4} & 0 & \mathbf{1/4} & 0 & 0 & 0 & \mathbf{1/4} \\ 0 & 0 & 0 & \mathbf{1/4} & 0 & 0 & 0 & \mathbf{1/4} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1/4} & 0 & \mathbf{1/4} & 0 & \mathbf{1/4} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1/4} & 0 & \mathbf{1/4} & 0 \end{bmatrix} \cdot \begin{bmatrix} u_{0,0} \\ u_{0,1} \\ u_{0,2} \\ u_{1,0} \\ u_{1,1} \\ u_{1,2} \\ u_{2,1} \\ u_{2,2} \\ u_{1,2} \end{bmatrix}^{(k)} \quad (3\text{-}14)$$

where the constant vector C equals to $-B/4$ in this example. Afterwards, the system is mapped to the memristor crossbar hardware.

Specifically, at iteration *k*, a new estimate of the unknown u vector is computed for the next iteration *k*+1 as:

$$u^{(k+1)} = C - R \cdot u^{(k)} \qquad (3\text{-}15)$$

where *R* is a modified coefficient matrix with the diagonal elements removed, and *C* is a constant vector that includes the boundary values. Eq. (3-15) can be implemented in a crossbar array by mapping *R* and *C* to the crossbar with numerical values represented by the memristor device conductance, as shown in Figure 3-3(a). By applying $u^{(k)}$ to the input rows of this crossbar as voltage pulses, the output currents collected at the columns represent the new estimated value of $u^{(k+1)}$. The process is then repeated iteratively by feeding $u^{(k+1)}$ to the system as the next input until desired accuracy is achieved.

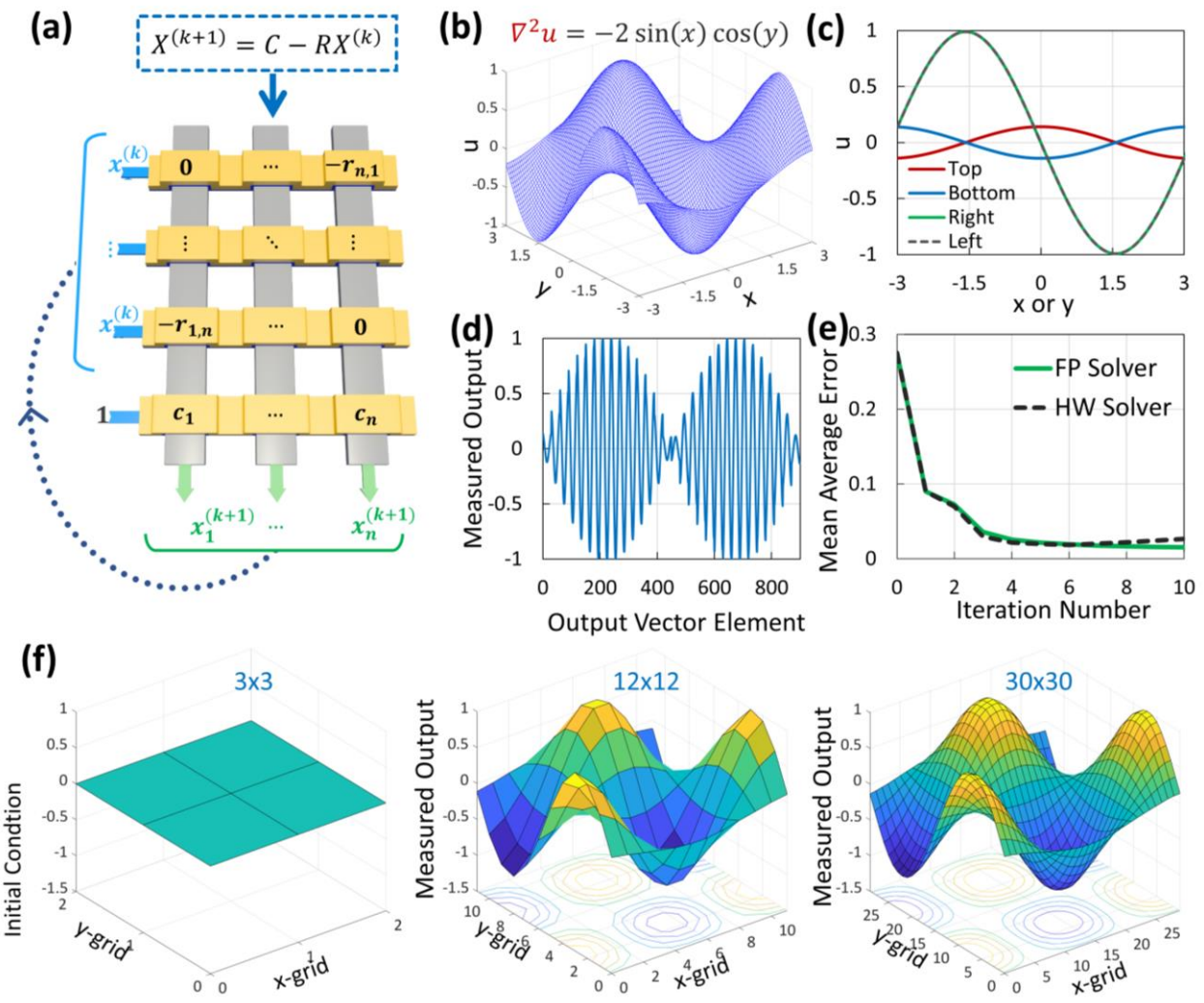

Figure 3-3: Experimental demonstration of solving a Poisson's equation. (a) Mapping the Jacobi method (used to iteratively solve Poisson's PDE) to a memristor crossbar-based system. A single crossbar is shown for illustration purpose. The solution is iteratively computed by applying the vector $\boldsymbol{u}^{(k)}$ as the row voltage of the crossbar and collecting the output currents at the columns which represent the numerical value of $\boldsymbol{u}^{(k+1)}$. (b) The Poisson's equation used as a test example, and a 3D plot of the intended solution. (c) The boundary conditions used in the example, measured at the four edges of the system. (d) Final measured output from the memristor-based PDE solver hardware, for the 900 grid points in the 30×30 mesh. (e) Evolution of the mean average error (MAE) for the memristor-based solver and a floating-point solver, measured against the exact numerical solution. A multi-grid technique was used during the iterations, where the system started with a 3×3

grid and ended with a 30×30 grid. (f) 3D reconstructions of the initial condition (with a 3×3 grid), and the measured outputs at iteration numbers 4 and 10, at grid sizes of 12×12 and 30×30, respectively.

3.2.3 Experimental Demonstration

The expected solution of the problem is shown in Figure 3-3(b), along with the boundary conditions shown in Figure 3-3(c). Eq. (3-7) is then converted to the matrix form using a five-point numerical stencil via the finite-difference method. Here we used a uniform grid, which typically results in a symmetric matrix with the non-zero elements along the penta-diagonal directions. In this example, only 4 elements along any row is non-zero after removal of the diagonal elements following the Jacobi method as shown in Figure 3-4. Specifically, when dividing the matrix with 3×3 slices (if the number of grid points are multiples of 3), only 4 different patterns are needed. We thus sliced the coefficient matrix into 3×3 patches and write the 4 patterns into a 16×3 array. Time multiplexing is then used to obtain the vector-matrix products from the crossbar output for different slices sharing the same pattern. The different partial-products are then summed through the board to obtain the final output. Note time multiplexing is not required in general, as parallel processing of the slices can be obtained if a larger crossbar hardware system can be built.

Using the proposed precision extension approach any target precision can potentially be achieved in the hardware system. Here, 16-bit precision is needed for the input and output vectors to achieve convergence and correct solutions. With the proposed approach, the Poisson's equation was iteratively solved using the memristor-based system at 16-bit precision. We utilized a simple coarse-to-fine grid approach to improve the numerical convergence speed, where we started with a 3×3 grid and ended with a 30×30 grid after ten system iterations for the same space. After each iteration, the solution is updated and the grid size is increased, where the coarse grid solution acts as an initial approximation for the next finer grid. These grids generate coefficient matrices of sizes

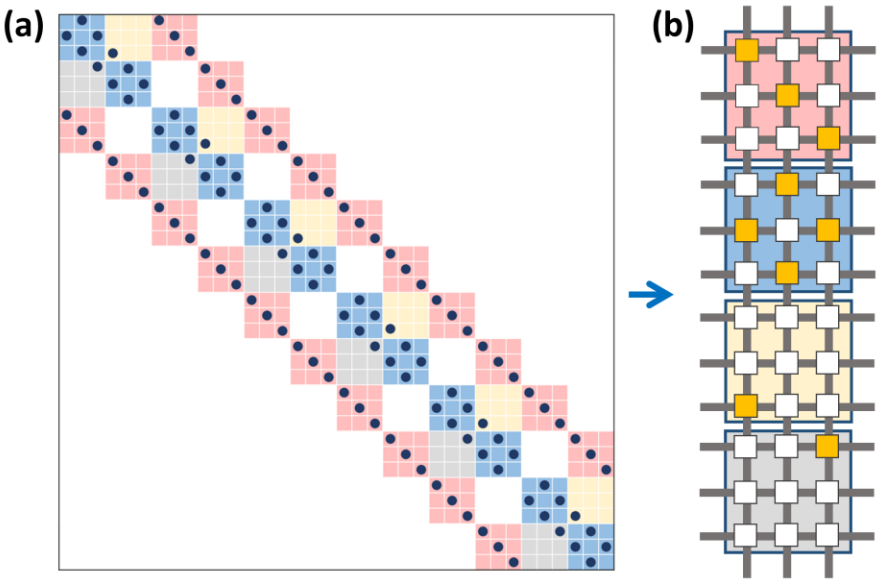

Figure 3-4: Slicing coefficient matrix. (a) An example of a 144×144 coefficient matrix (from a 12×12 grid) generated using FDM and sliced into 3×3 slices. The diagonal of the coefficient matrix is removed following the Jacobi method. The active slides (containing nonzero elements) are highlighted with different colors. (b) The four different slice patterns of the matrix can be mapped to 4 3×3 arrays in a 12×3 crossbar array through time multiplexing.

ranging from 81 to 8.1×10^5 elements, while the number of active nonzero slices ranges from 7 to 1.42×10^3.

The output of the memristor crossbar system for the final 30×30 grid points is shown in Figure 3-3(d). The measured output was compared with the exact solution obtained using the inverse matrix technique for the same equation, and the mean absolute error (MAE) in Eq. (3-16) was measured and plotted against the iteration number, as shown in Figure 3-3(e). For comparison, results obtained from a standard floating-point solver are also plotted.

$$\mathrm{MAE} = \frac{\sum_{j=0}^{n-1}\left|x_j^E - x_j^N\right|}{n} \quad (3.16)$$

where x^E is the exact solution of the problem, x^N is the numerically computed solution either using the memristor PDE solver or a floating-point solver, and n is the number of grid points.

The results show that both solutions converge at roughly the same rate with similar error figures. Using the multi-grid approach, 10 iterations were enough for the hardware system to achieve an absolute error below 2.7% compared with the exact solutions. Figure 3-3(f) shows three-dimensional reconstructions of the experimentally obtained solution from the memristor-based solver, at different iteration numbers. The solution after 10 iterations shows excellent match with the expected solution. Note the small differences in the results obtained from the memristor-based solver and the floating-point solver in Figure 3-3(e) are due to the device variability in the hardware system, as the precision-extension technique was only applied to the input vectors and devices in this experiment. By applying the precision-extension technique to the output as well (by ADC quantization), a precise match between the memristor-based solver and the floating-point solver can be obtained as shown in Figure 3-5.

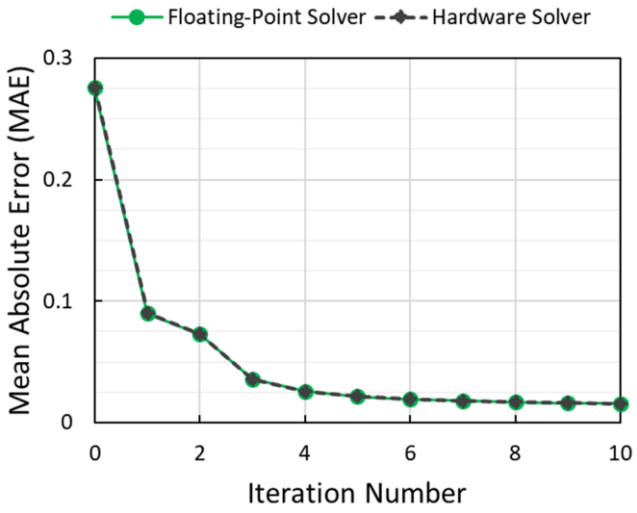

Figure 3-5: Comparison of results obtained from the crossbar PDE solver and a floating-point solver, after applying precision extension technique including ADC quantization for the 2D Poisson's equation test case. The mean absolute errors were calculated against the exact numerical solution (through the inverse matrix technique).

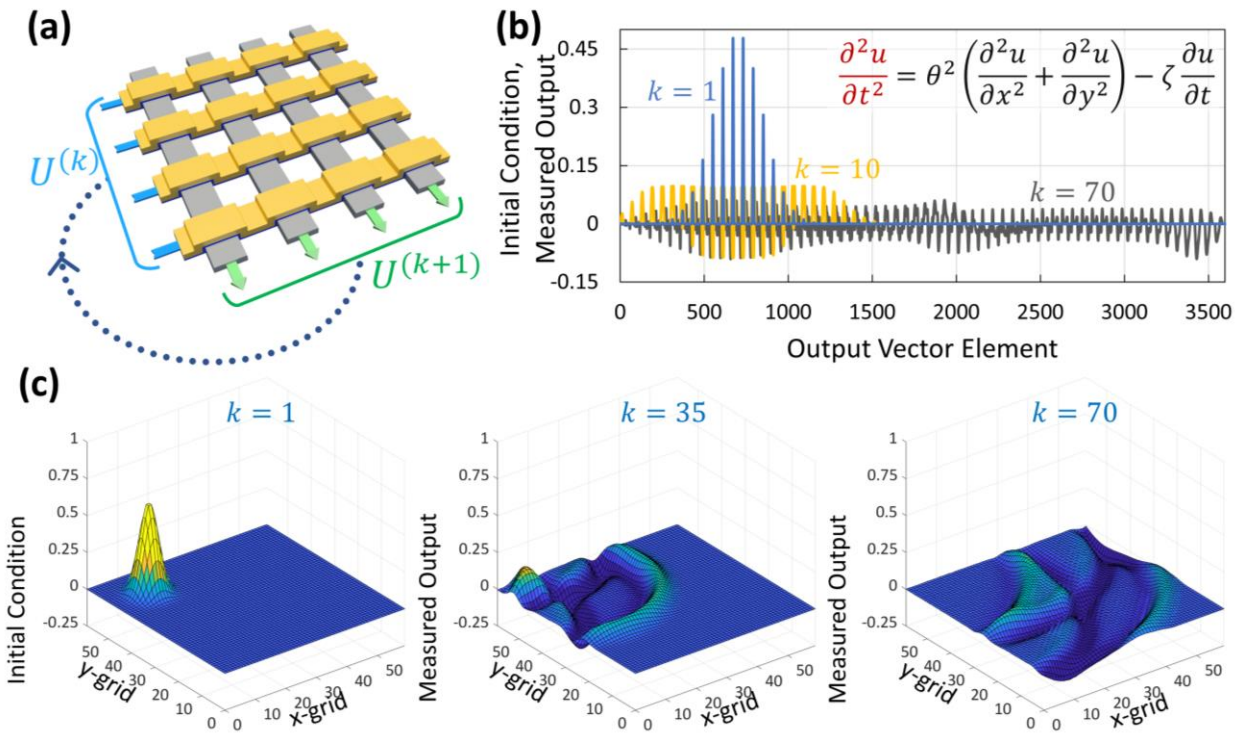

Figure 3-6: Experimental demonstration of solving a damped 2D wave equation (a) A general approach to solve time-evolving PDEs using memristor crossbar arrays. The output currents represent a new estimation for the next time step. (b) The initial condition (at $k = 1$) and the measured outputs of the 10th and 70th iterations for a damped wave equation PDE test case (inset). The test problem is iteratively solved in a 60×60 grid. (c) 3D reconstructions of the initial condition, showing a droplet touching the water surface with a gaussian-shaped surface profile, and the measured outputs at iteration numbers 35 and 70, where the z-axis represents the water surface level.

3.3 Time Evolving PDE example

The second set of PDEs we tested using the memristor-based hardware system are time evolving problems. In this case, the PDE includes partial derivatives with respect to time along with other variables. Typically, numerical methods such as finite-difference are used to map the equation to the matrix form [80], and the new state of the system is computed in an iterative manner. At the hardware level, this process is again reduced into a series of VMM operations, where the

output of the crossbar array at one time frame is used as the input for the next iteration, as shown in Figure 3-6(a).

As an example, we experimentally solved a two-dimensional wave equation using the memristor-based setup. The equation is:

$$\frac{\partial^2 u}{\partial t^2} = \theta^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \zeta \frac{\partial u}{\partial t} \quad (3\text{-}17)$$

, where u is the wave amplitude, θ is the wave speed and ζ is a decay constant. This equation represents a classical physics description of the propagation of two-dimensional waves and can be utilized to visualize shallow water surface in a computationally inexpensive manner [81].

3.3.1 2D Wave PDE Mapping

To map the 2D wave system described in Eq. (3-17) to the hardware setup, we first discretized it following FDM as:

$$\frac{u_{i,j}^{(t+1)} - 2u_{i,j}^{(t)} + u_{i,j}^{(t-1)}}{(\Delta t)^2} = \theta^2 \left(\frac{u_{i+1,j}^{(t)} + u_{i,j+1}^{(t)} - 4u_{i,j}^{(t)} + u_{i-1,j}^{(t)} + u_{i,j-1}^{(t)}}{h^2}\right) - \zeta \left(\frac{u_{i,j}^{(t)} - u_{i,j}^{(t-1)}}{\Delta t}\right) \quad (3\text{-}18)$$

where u represents the wave height, i is the *x*-axis grid index, j is the *y*-axis grid index, h is the distance between two neighboring grid points along the *x*-axis or *y*-axis, t is the time index, Δt is the numerical time step, θ is the wave speed and ζ is the decay (damping) constant. Here, central FDM is used for the second-order partial derivatives while backward FDM is used for the first-order partial derivative. Eq. (3-18) can be re-written in a five-point stencil format as:

$$u_{i,j}^{(t+1)} = (2 - \zeta\Delta t)u_{i,j}^{(t)} + (\zeta\Delta t - 1)u_{i,j}^{(t-1)} + \left(\frac{\theta\Delta t}{h}\right)^2 \left(u_{i+1,j}^{(t)} + u_{i,j+1}^{(t)} - 4u_{i,j}^{(t)} + u_{i-1,j}^{(t)} + u_{i,j-1}^{(t)}\right) \quad (3\text{-}19)$$

and mapped to a matrix form as (shown for a 3×3 grid for illustration purpose):

$$\begin{bmatrix} u_{0,0} \\ u_{0,1} \\ u_{0,2} \\ u_{1,0} \\ u_{1,1} \\ u_{1,2} \\ u_{2,1} \\ u_{2,2} \\ u_{1,2} \end{bmatrix}^{(t+1)} = \alpha_1 \begin{bmatrix} u_{0,0} \\ u_{0,1} \\ u_{0,2} \\ u_{1,0} \\ u_{1,1} \\ u_{1,2} \\ u_{2,1} \\ u_{2,2} \\ u_{1,2} \end{bmatrix}^{(t)} + \alpha_2 \begin{bmatrix} u_{0,0} \\ u_{0,1} \\ u_{0,2} \\ u_{1,0} \\ u_{1,1} \\ u_{1,2} \\ u_{2,1} \\ u_{2,2} \\ u_{1,2} \end{bmatrix}^{(t-1)} + \alpha_3 \begin{bmatrix} \mathbf{-4} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ \mathbf{1} & \mathbf{-4} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{-4} & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & \mathbf{-4} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{-4} & \mathbf{1} & 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{-4} & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & \mathbf{-4} & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{-4} & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{-4} \end{bmatrix} \cdot \begin{bmatrix} u_{0,0} \\ u_{0,1} \\ u_{0,2} \\ u_{1,0} \\ u_{1,1} \\ u_{1,2} \\ u_{2,1} \\ u_{2,2} \\ u_{1,2} \end{bmatrix}^{(t)} \quad (3\text{-}20)$$

, where the constants $\alpha_1 = 2 - \zeta\Delta t$, $\alpha_2 = \zeta\Delta t - 1$, and $\alpha_3 = (\theta\Delta t/h)^2$. Similar to the Poisson's equation case, the number of nonzero elements in each row equals to the stencil size (5), while the row length equals to the total number of grid points. This results in highly sparse coefficient matrices.

3.3.2 Experimental Demonstration

We solved the wave equation in a 60×60 grid, with a wave speed of $\sqrt{0.37}$, a decay constant of 2.5×10^{-2}, spatial steps $h_x = h_y = 0.1$, and time step $\Delta t = 0.1$. Using the finite-difference method, Eq. (3-17) is re-written as:

$$U^{(k+1)} = \alpha_1 U^{(k)} + \alpha_2 U^{(k-1)} + \alpha_3\left(A \cdot U^{(k)}\right) \qquad (3\text{-}21)$$

where U is the targeted solution vector, k is the iteration number, $\alpha_1, \alpha_2, \alpha_3$ are constants based on θ, ζ, h_x, h_y and Δt, and A is the coefficients matrix. Using a five-point stencil to generate the coefficient matrix, the matrix A contains 1.3×10^7 elements but only less than 0.14% of the elements are nonzero. After removing the diagonal elements, the sparse coefficient matrix is divided into 3×3 patches with a total of 5840 active slices using the software package. These slices follow 4 different patterns which are then mapped directly to 4 3×3 crossbars in the 16×3 crossbar as shown in Figure 3-4. Similar to the static PDE example, time-multiplexing was used to perform VMM operations on the 3×3 crossbars for slices sharing the same pattern.

As an initial condition, we set $U^{(1)}$ to be a Gaussian shape representing a droplet touching the water surface. The water droplet initiates the two-dimensional wave, and iterative operations were performed through the memristor-based system to solve the evolution of the water wave. The input and output vectors are encoded as 16-bit numbers. Precision extension techniques were also applied to the matrix coefficients to reduce error propagation in the time evolving iterations. We ran the process for 70 successive iterations to solve the wave propagation through the water pool and its reflection from the pool edges. The initial input vector to the system at $k = 1$ and the measured outputs at $k = 35$ and 70 are shown in Figure 3-6(b). The 3-dimensional reconstructions of the solution from the experimentally measured output of the memristor-based hardware system are in Figure 3-6(c), showing a snapshot of the wave propagation at different times, and verify the system's ability to successfully solve this time varying problem. More examples of the solutions are in Figure 3-7.

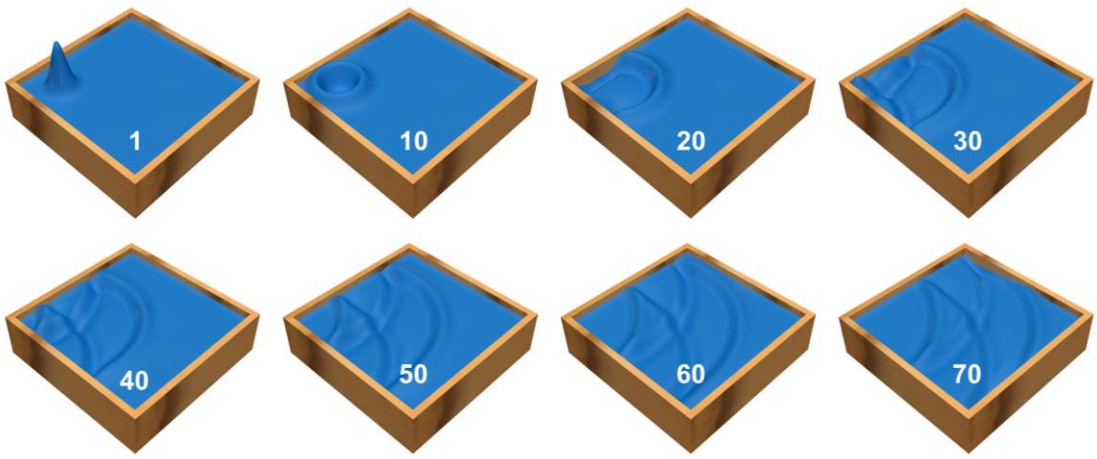

Figure 3-7: Additional examples of 3D reconstructed solutions for the damped wave system test case at different iterations, experimentally obtained from the crossbar-based PDE solver hardware system.

3.4 Conclusion

In this study, we showed that memristor-based in-memory computing systems can be used to process tasks requiring high precision and accurate solutions, beyond what has been demonstrated for soft computing tasks where high device variability may be tolerated. Despite the limited native precision offered by the devices, architecture-level optimizations such as the precision-extension techniques proposed here can effectively lead to computations achieving 64-bit accuracy. We experimentally demonstrated a high-precision memristor crossbar-based PDE solver. Using a tantalum-oxide memristor crossbar, we successfully solved elliptic and hyperbolic PDE equations representing static and time-evolving systems, for the widely used Poisson's equation and a classical water wave propagation problem, respectively.

Our studies showed that challenges including device variability, limited equivalent precision, and limited on/off ratio can be successfully addressed even for high-precision computing tasks. Additionally, the precision extension approach utilized in our approach allows the system to be able to dynamically adjust the system precision as needed, and thus can efficiently allocate hardware resources depending on the task requirement on hand. As a result, a common hardware platform may be used to process different tasks, including both soft-computing and hard-computing problems. We believe such demonstrations, showing memristor crossbars can be used to directly solve high-precision computing tasks (instead of playing a supporting role to a digital system), significantly broaden the appeal of memristor-based hardware systems, and pave the way for the development of more general-purpose, memristor-based computing systems.

We anticipate a fully integrated computing system based on arrays of memristor crossbars monolithically integrated on complementary metal-oxide-semiconductor supporting circuitry [82] can offer a scalable computing system with very high processing speeds and power efficiency,

owing to its ability to natively compute information in-memory and to its high level of parallelism. And the proposed memristor-based computing hardware system is well positioned for soft as well as hard data-intensive computing tasks now and in the future.

Chapter 4. Second-order Memristor Device and Network Applications

As discussed in the previous 2 and 3 chapters, memristor networks have already shown great promise in applications such as numerical computing, clustering and classification, with the functionality and the system size rapidly growing in recent years [36], [78], [83]. On the other hand, networks based on first-order devices typically lack the ability to directly process temporal data, where information is encoded in the relative timing of the input signals. Below we discuss how second-order memristor devices can natively respond to the timing information of the inputs, and how networks based on second-order memristor devices can be used to efficiently process temporal data without expensive pre-processing processes.

4.1 Encoding of Timing Information

During the RS process, the internal ion configuration in the switching layer is modified by ionic drift and diffusion, which are affected by an applied electric field and local temperature through the Joule heating effect. Specifically, the rise and decay of the local temperature T can occur at a much shorter time scale than the ionic processes, providing a second internal state variable and short-term dynamic processes required by a second-order memristor [42]. Indeed, this effect was verified by us in a study on a tantalum oxide-based memristive device [28], [41], [42].

The impact of the local temperature was first verified from timing-controlled pulse experiments [42]. After a single reset pulse (V_{RESET} = 1.4V, t_{RESET} = 40μs), the conductance change following 100 consecutive set pulses (V_{SET} = -0.9V) was traced for different set pulse intervals, $t_{interval}$, and set pulse duration, t_{SET}, as shown in Figure 4-1(a) [42]. Two regimes with distinct

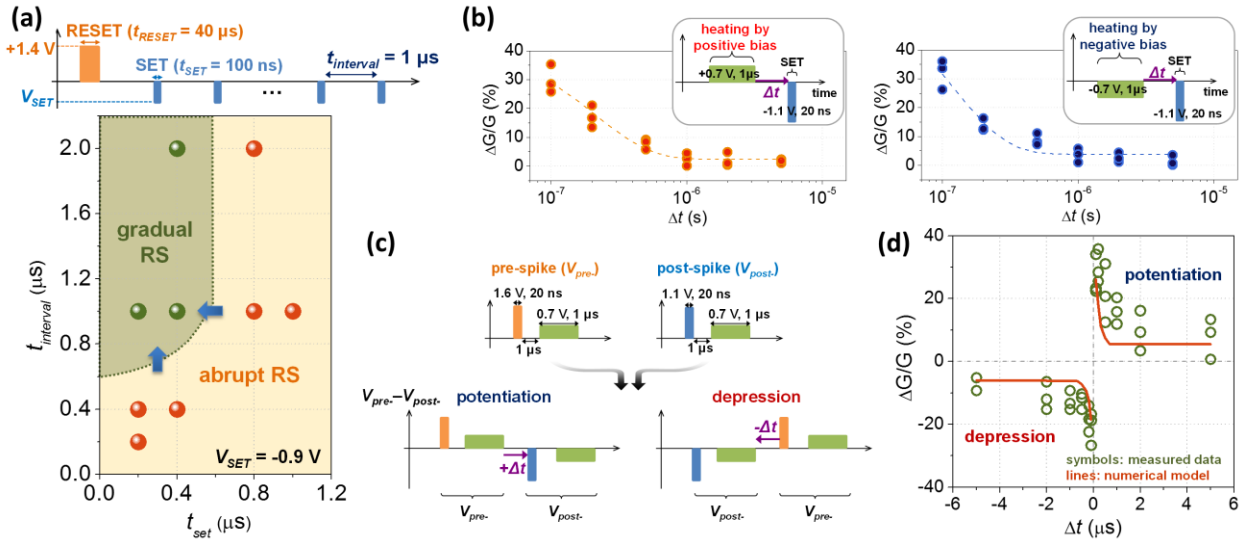

Figure 4-1: Second-order memristor effects. (a) Two different types of conductance changes, depending on the pulse interval and pulse duration. (b) Controlling the relative timing between a heating pulse and a programming pulse leads to controlled conductance changes, where the relative timing information is encoded by the short-term dynamics of the internal temperature. (c) Configuration of the pre-synaptic and post-synaptic spikes, each consisting of a (high and narrow) programming element followed by a (low and long) heating element. The relative timing of the pre- and post-spikes determines the potentiation or depression of the device, and naturally lead to effects such as STDP (d), showing the measured conductance changes (dots) and simulation results based on a second-order memristor model (solid lines). (Images courtesy of [42].)

resistive switching (RS) behaviors can be clearly separated: abrupt RS at short $t_{interval}$ and long t_{SET}, and gradual RS at long $t_{interval}$ and short t_{SET}. This effect can be explained by the temperature dynamics inside the device: Joule heating generated by a set pulse elevates the local temperature T inside the device and the higher T can exponentially speed up V_Os migration. As a result, when $t_{interval}$ is short enough, the cumulative effects from consecutive set pulses creates a high enough temperature that leads to rapid ion movement and abrupt conductance modulation. The cumulative temperature rise is less pronounced at long $t_{interval}$ due to the temperature decay from spontaneous

heat dissipation after the set pulse removal, and the lower cumulative temperature in these conditions leads to more gradual conductance modulations. Similar effects can be obtained by controlling the width of the programming pulses. These results verify the internal temperature dynamics (represented by state-variable *T*) can indeed affect the evolution of the conduction channel (represented by state-variable *w*) and can be used to construct a second-order memristor device.

The role of the temperature as a second state variable is more clearly revealed by separating the effects of heating and programming, as shown in Figure 4-1(b). Here a first pulse (named heating pulse, 0.7V, 1μs) is sufficiently long to create a sizable temperature change but the voltage amplitude is sufficiently low to not able to directly initiate ion migration; a second pulse (named programming pulse, -1.1V, 20ns) provides high enough voltage but the pulse duration is too short to cause measurable conductance change by itself either. Figure 4-1(b) shows conductance changes as a function of the interval of the two pulses, Δt. With short Δt, the cumulatively increased local temperature *T* can cause sufficient ion migration during the programming pulse and lead to measurable conductance changes. Due to the spontaneous temperature decay between the pulses, the extent of the temperature increase during the programming pulse is thus a function of the interval between the two pulses Δt. Indeed, the conductance modulation obtained after the programming pulse decreases as Δt increases and the effect becomes negligible for $\Delta t \sim 0.5\mu s$. In addition, the effect of the heating pulse is independent of the voltage polarity (Figure 4-1(b)), verifying the Joule heating role played by this pulse. These results unambiguously verify that the internal short-term dynamics (provided by *T* here) can be used to code the timing information of the inputs, and internal variables such as *T* can be used as a second-order state-variable to emulate

the short-term Ca^{2+} dynamics in biology to faithfully implement timing- and rate-based learning rules.

An example of native implementation of important timing-based learning rule – spike-timing dependent plasticity (STDP) using a second-order memristor device is shown in Figure 4-1(d). As in biological systems, the pre-synaptic and post-synaptic spikes are simple and non-overlapping. The nearly identical pre- and post-spikes are applied to the top and bottom electrode, respectively. Each spike consists of a short, programming element and a long, heating element, as shown in Figure 4-1(c). When the pre-synaptic spike arrives before the post-synaptic spike ($\Delta t > 0$, Figure 4-1(d) left), residue of the temporal temperature increase caused by the pre-synaptic spike facilitates the programming element in the post-synaptic spike and causes conductance increase (i.e. potentiation), with shorter Δt leading to a stronger potentiation effect. Similarly, in the opposite case, when the post-synaptic spike arrives before the pre-spike ($\Delta t < 0$, Figure 4-1(d) left), the temporal temperature rise from the post-synaptic spike strengthens the depression effect in the pre-spike, with shorter Δt leading to a stronger depression. By taking advantage of the internal short-term dynamics, the second-order device can thus naturally implement STDP with desired polarity and timing-dependence, using only simple, non-overlapping pulses. Similarly, rate-dependent plasticity behaviors can also be successfully implemented using internal temperature as the second-state variable to encode the timing information, using the same spike designs [42]. Other devices using other parameters as the second state-variable, e.g. ion mobility, have also been successfully demonstrated [51].

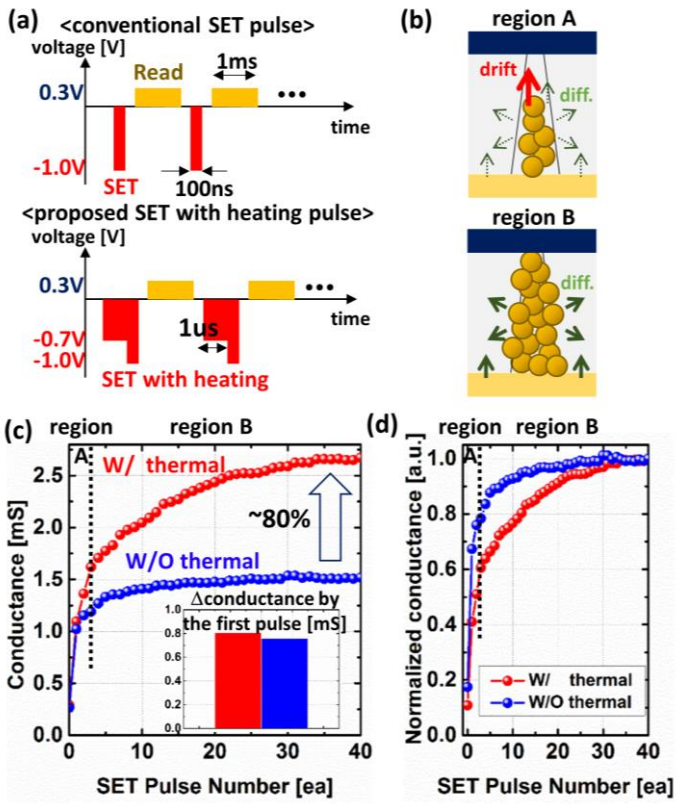

Figure 4-2: Device optimization utilizing multiple state variables. (a) Schematic of the pulse, including a heating element and a programming element. (b) Dominant ion migration processes in region A and region B of the RS process. (c) The proposed approach can selectively enhance processes in region B and achieved a significant improvement in the analog RS dynamic range. (d) Data in (c) after normalization of the conductance, highlighting the improvement in incremental conductance change by utilizing the temperature state variable.

4.2 Device Optimization Using Multiple State-Variables

Before introducing temporal data processing applications in second-order memristor networks, we show that the understanding of the dynamics of different internal state variables can also leads to new approaches to optimize the device operation, such as enhancing the dynamic range of the analog RS behavior [84]. Figure 4-2 shows the operation of a Ta_2O_{5-x}-based memristor device in the analog RS regime. The device behavior can be roughly divided into two regions depending on the relative amount of conductance change: region A with abrupt increase, followed

by region B with a much more gradual conductance change. The different behaviors can be understood by carefully examining the different physical processes involved in RS. Specifically, before the formation of the complete filament (Figure 4-2(b), top), a high electric field exists and the V_Os drift process dominates the V_Os migration, which in turn promotes the conductive filament (CF) length growth along the E-field direction resulting in very non-linear conductance changes, corresponding to region A. After the CF is completed (Figure 4-2(b), bottom), V_Os drift is suppressed due to the reduced E-field. On the other hand, more pronounced temperature increase can occur due to the increased current through the device and thus resulting in increased Joule heating effect. The elevated temperature enhances V_Os diffusion which expands the cross-sectional area of the CF, resulting in more gradual and linear conductance changes, corresponding to region B. Thus, an approach to increase the analog RS dynamic range should selectively enhance the V_Os diffusion process with respect to the V_Os drift process. These effects can be further described with Eq. (4-1) and Eq. (4-2) below, where a simple 1D rigid point ion hopping model by Mott and Gurney is employed to describe the ion diffusivity D and drift velocity v, respectively [41], [85].

$$D = \frac{1}{2}a^2 f exp\left(-\frac{E_a}{kT}\right) \qquad (4\text{-}1)$$

$$v = af exp\left(-\frac{E_a}{kT}\right)\sinh\left(\frac{qaE}{kT}\right) \qquad (4\text{-}2)$$

Here, f is the attempt-to-escape frequency, E_a is the activation energy for the hopping process, and a is the effective hopping distance. T is the local temperature discussed earlier, and E is the local electric field. Since the diffusion process is primarily affected by the local temperature (via Joule heating) and not directly affected by the electric field (Eq. (4-1)), while the drift process is primarily affected by the electric field (although also affected by local temperature but to a lesser extent due to the different factors in the T term, Eq. (4-2)), increasing the programming pulse amplitude will primarily result in the enhancement of the drift process and will not necessarily

improve the analog behavior. Instead, a more ideal approach is to decouple the local temperature effect from the programming pulse. This effect can be implemented by separating the heating effect from the field effect, e.g. by using programming pulses consisting of a separate heating element and a programming element, as shown in Figure 4-2(a). Indeed, this approach significantly improves the analog RS dynamic range by a factor of ~80%, as shown in Figure 4-2(c)-(d). Note that even though temperature increase can exponentially increase both drift and diffusion processes, the heating effect is more pronounced in region B due to the higher current. As a result, it selectively enhances the processes in region B (where V_Os diffusion plays a larger role) and results in a larger, more linear analog RS region, by taking advantage of the dynamics of the different internal state variables.

4.2.1 Simulation Results

The explanation of the thermal effect in selectively improving and extending the analog behaviors is further quantitatively supported by detailed multi-physics numerical simulation by self-consistently solving the electronic and ionic current equations [40], [41]. Specifically, three partial differential equations (PDEs): (1) drift/diffusion continuity equation for *Vo*s, (2) current continuity equation for electron conduction, and (3) Fourier equation for Joule heating are considered and solved self-consistently to obtain the filament size state variable *w* and the temperature variable *T* during the RS process [40], [41]. Figure 4-3(a) shows a 2-D cross-section map obtained from the model showing the temperature distribution at the peak of the SET pulse for different conditions. As the filament approaches completion, the initially localized heat generation near the gap area in region A is enhanced and spreads out to the entire filament region in region B. In addition, implementing the heating element further increases the local temperature. This effect is more evident in Figure 4-3(b), which plots the temperature evolution at the marked

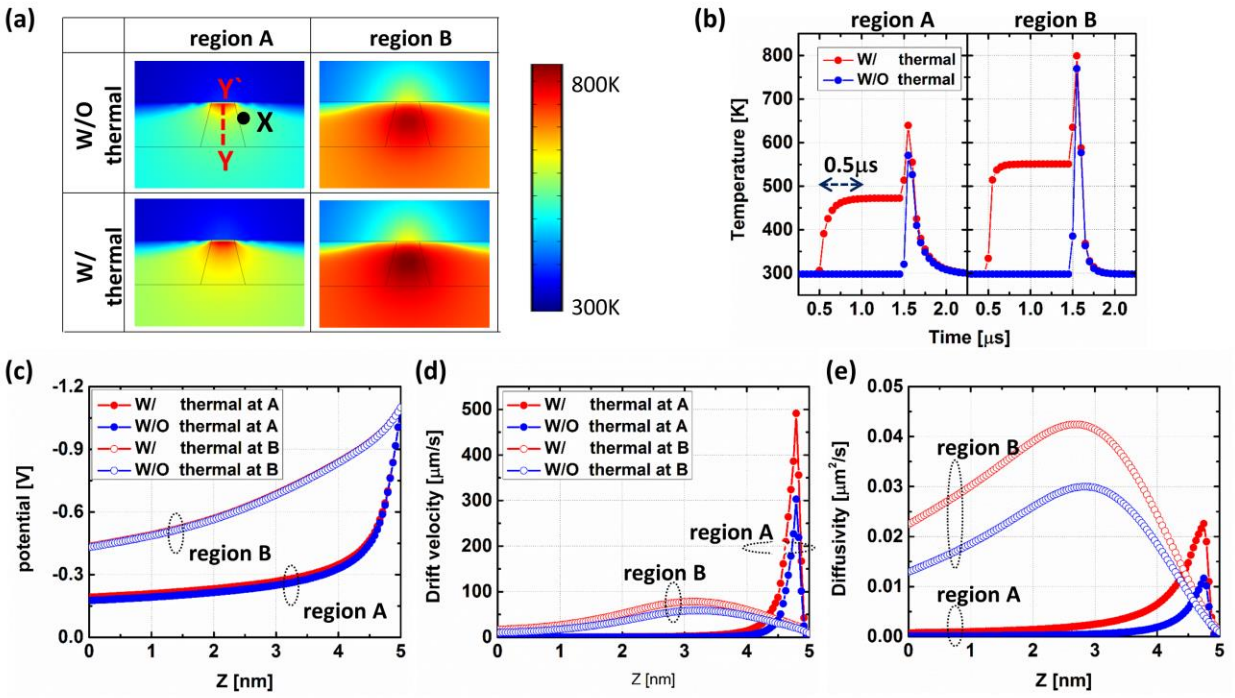

Figure 4-3: Numerical simulation on the heating pulse effect. (a) Simulated temperature distribution in the device at different points during filament growth, for both programming schemes. (b) Transient temperature data of the two cases. The temperature was recorded for location X in (a). A time constant of ~ 0.5μs was observed for the device to reach thermal steady state. (c) Electrical potential, (d) Drift velocity, and (e) Diffusivity distribution along the Y'-Y line in (a) for different stages during the filament growth. The simulation results show that drift velocity decreases and diffusivity increase from region A to region B. The increase in local temperature through the heating pulse enhances diffusivity in region B and improves the analog RS behavior.

location X in Figure 4-3(a). Other physical parameters such as the electrical potential, the *Vo*s drift velocity, and the *Vo*s diffusivity are also extracted along the YY` line marked in Figure 4-3(a) and shown in Figure 4-3(c)-(e), respectively. First, by comparing region A and region B in Figure 4-3(d)-(e), we note that the drift process is indeed dominating in region A (Figure. 4-3(d)), as expected. As the filament grows and the device enters region B, the diffusion process is significantly enhanced (Figure 4-3(e)), as discussed earlier. Furthermore, when the heating

element is applied a larger enhancement is observed in the diffusion component (Figure 4-3(e)) compared with enhancement in the drift component (Figure 4-3(d)) at the same conditions, since the temperature increase is less pronounced in region A due to the incomplete filament. These simulation results clearly reveal the different roles that diffusion and drift processes play during the filament growth process and support the experimental observations that diffusion can be selectively enhanced during the filament expansion stage (region B) to achieve better analog RS behavior. From the temperature map in Figure 4-3(a), we can also point out that most of the heat generated by Joule heating dissipates through the TE due to its high thermal conductivity. As a result, controlling the thermal leakage path by engineering the TE material can potentially improve the thermal effect even further.

4.3 Second-order Memristor Network Application

The ability to naturally encode temporal data in second-order memristors in the section 4.1 suggests development of a network based on such devices can efficiently process temporal spiking inputs. Below we discuss a possible implementation of STDP-based spiking neural network using second-order memristor devices for temporal information processing [86].

The proposed crossbar network is shown in Figure 4-4(a). Voltage pulses, representing spiking inputs, are applied to the presynaptic side (at each row). The spikes produce current flow through the network, and the charges are collected and integrated at the postsynaptic neurons (at each column). Once the post-neuron's potential reaches a threshold and it will fire and create a post-spike that also back-propagates through the network. The firing is thus asynchronous. A neuron firing event also resets all output neurons' membrane potential to the resting value. The pre- and post-synaptic spikes cause memristor devices in the network to undergo either potentiation by correlated events, i.e., pre-synaptic spike arriving before post-synaptic spike, or

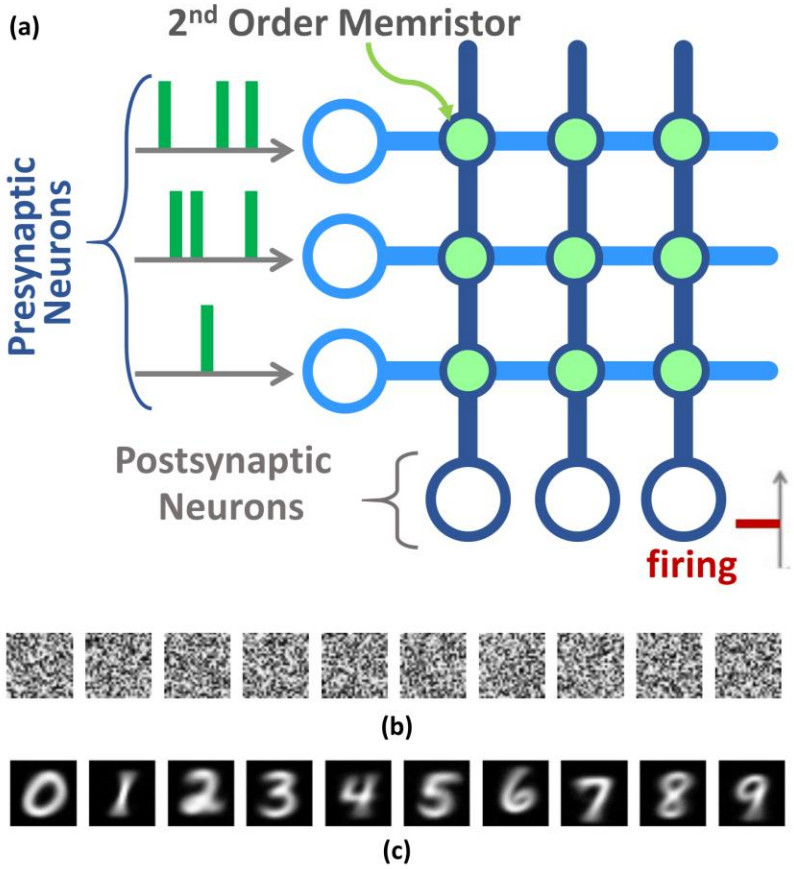

Figure 4-4: Schematic of the spiking neural network and example of training using the MNIST dataset. (a) Schematic of the spiking network based on second-order memristors. The device conductance modulation (training) is based purely through the asynchronous firing of the pre- and post-synaptic neurons. (b-c) Example of training using the MNIST dataset, showing (b) the randomly distributed initial weights before training, and (c) features evolved in the network after unsupervised learning. (Images courtesy of [86].)

depression by uncorrelated events, i.e., post-synaptic spike arriving before pre-synaptic spike, resembling the STDP rule [44], [46].

To verify the learning ability of the spiking neural network based on the second-order RS effect, a classical hand-written digit database, i.e. MNIST database, was used to train the network in an unsupervised fashion. The network was implemented in a 784×10 crossbar array and its

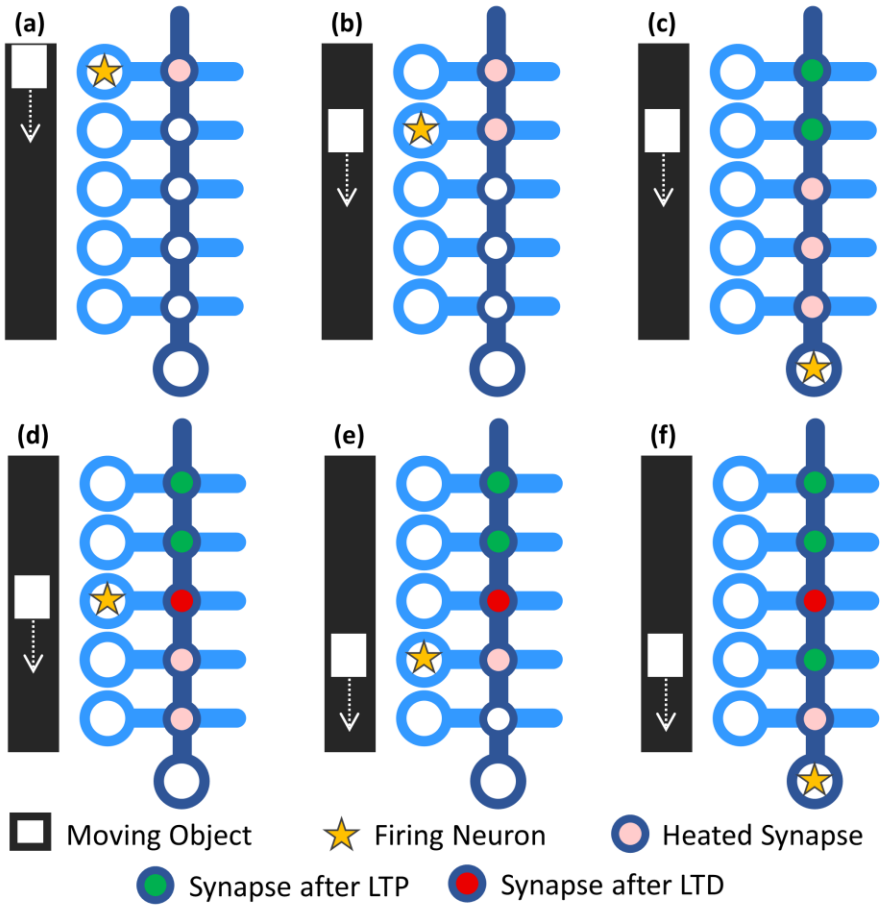

Figure 4-5: Schematics showing how the internal dynamics of the second-order device-based network can process temporal data. A spike from the input creates current flow and also a temperature trace as the object moves. When combined with the backpropagating spike from the output neuron, either potentiation or depression in the devices can be initiated, depending on the relative timing of the pre- and post-spikes. (Images courtesy of [86].)

operation was analyzed through simulation based on the second-order memristor model [86]. Starting from a random initial condition shown in Figure 4-4(b), desired features characteristic of the10 classes of inputs can be successfully obtained purely through the native STDP processes in the second-order memristor network, as shown in Figure 4-4(c).

After verifying the basic learning ability of the network as shown in Figure 4-4, the capability of learning and processing temporal features is then tested using video inputs. Simple videos of an object moving along one direction were created for the training process, as shown in Figure 4-5. During each frame, the moving object, represented by a white pixel over a black background, sends a spike to the input neuron corresponding to the object's location, shown in Figure 4-5. As a result, the spike creates a current flow through the devices in the corresponding row and elevates the internal temperature T of those devices, which will undergo a spontaneous decay after removal of the spike as the object moves to the next location. This process creates a temperature profile along the object's direction, effectively representing a shadow of the passed object (Figure 4-5(a) and 4-5(b)). When one of the post-synaptic neurons accumulates enough charges and fires, the backpropagating post-synaptic spike can initiate potentiation (since the post-spike is after the pre-spike) on cells that still possess sufficient residue temperature (Figure 4-5(c)). Similarly, the spike from the post-neuron also leaves a temperature trace on the cells and can cause depression on the devices that receives the spike from the moving object at a later time (Figure 4-5(d)). Since this effect is only valid in a few frames before fully vanishing of the decaying temperature traces caused by the pre- or post-spike, alternative potentiation and depression patterns will develop in the network and each pattern corresponds to a specific speed of the moving object (Figure 4-5(e) and 4-5(f)).

The proposed learning mechanism was tested through simulation in a 128×7 crossbar network corresponding to 128 locations in the video frame and 7 dictionary elements. After initialization of the network with random weights as shown in Figure 4-6(a), each dictionary element learned a different speed using the approach discussed above, represented with clearly distinguishable unique patterns, as shown in Figure 4-6(b)-(d). Then, the patterns can function as

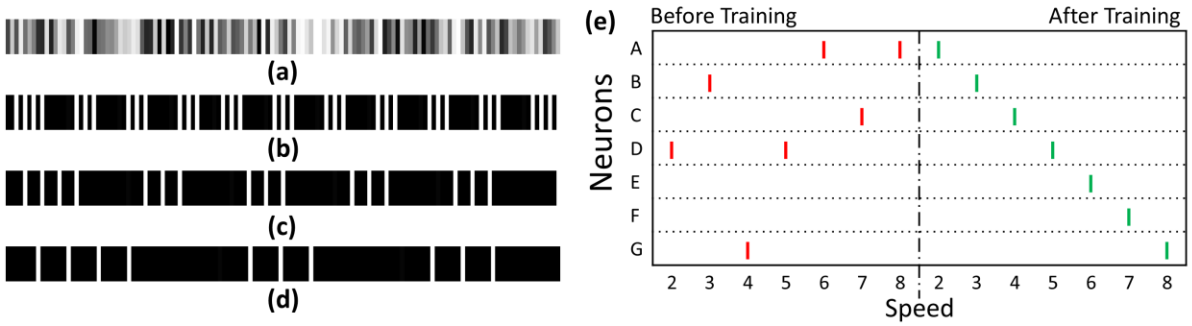

Figure 4-6: Simulation results showing a second-order memristor network used to process video of a moving object. (a) Initial random weight pattern. (b)-(d) Learned patterns from three different neurons corresponding to three moving speeds of the object. (e) Classification of the object's speed from the video, showing results obtained before training and after training. (Images courtesy of [86].)

fingerprints of the object's moving speeds in a subsequent classification stage. As shown in Figure 4-6(e), before training the network was not able to classify the input speed (based on the winning neuron), with no correlation of the winning neuron number and the speed. After training the network the object speed (and direction) can be correctly classified from the video. In this example, both the first-order and the second-order state variables (corresponding to conductance and temperature changes, respectively) are needed to naturally implement the STDP-like rule and encode and process the input video data.

4.5 Conclusion

While networks based on conventional first-order memristors already exhibit excellent potential for tasks such as data clustering and numerical computation through their ability to perform matrix operations, as discussed in chapters 2 and 3, second-order memristor devices can naturally encode temporal data and can lead to dynamic networks that are capable of effective processing of temporal inputs. The different internal state variables and their associated dynamic

processes during RS in turn provide a number of exciting opportunities: to naturally mimic biological processes, to provide new approaches for device optimization, and to natively encode and process temporal information. These studies, along with continued fundamental device- and material-level understanding and optimizations combined with algorithm and architecture-level developments, can hopefully lead to future highly-efficient bio-inspired computing systems.

Chapter 5. Fabrication of a Practical Size of Memristor Network

Up to Chapter 4, we introduced several studies on first-order and second-order memristor devices and networks, focusing on approaches inspired by biology. The crossbar network can efficiently implement bio-inspired computing algorithms due to the co-location of memory and computation with high level of parallelism. However, the maximum size of arrays used in the previous chapters remains relatively small at 16×3, and data processing for more complex algorithm demands larger sizes of memristor arrays to fully take advantage of its parallelism. For example, convolutional neural networks (CNNs) require massive amounts of VMM operations during both training and testing, and can be dramatically speeded up in a hardware system based on larger memristor arrays. In this chapter, we discuss factors that can affect the practical size of memristor arrays from the device fabrication point of view.

5.1 Forming-free Device for High-yield of Memristor Array

We showed the concept of the slicing technique in chapter 3 to extend the native precision of memristor devices to express longer bit lengths, while facilitating the usage of smaller memristor arrays. In addition, several previous studies on suitable size of memristor arrays have reported that 16×16 or 32×32 is a practical array size as a basic unit of memristor-based processor, e.g. memory-processing-unit (MPU). Larger array may enhance the network performance with higher parallelism, but may not always be the best option, since the essential component, analog-to-digital converter (ADC), to convert the analog current signal to be sent to the rest of the system

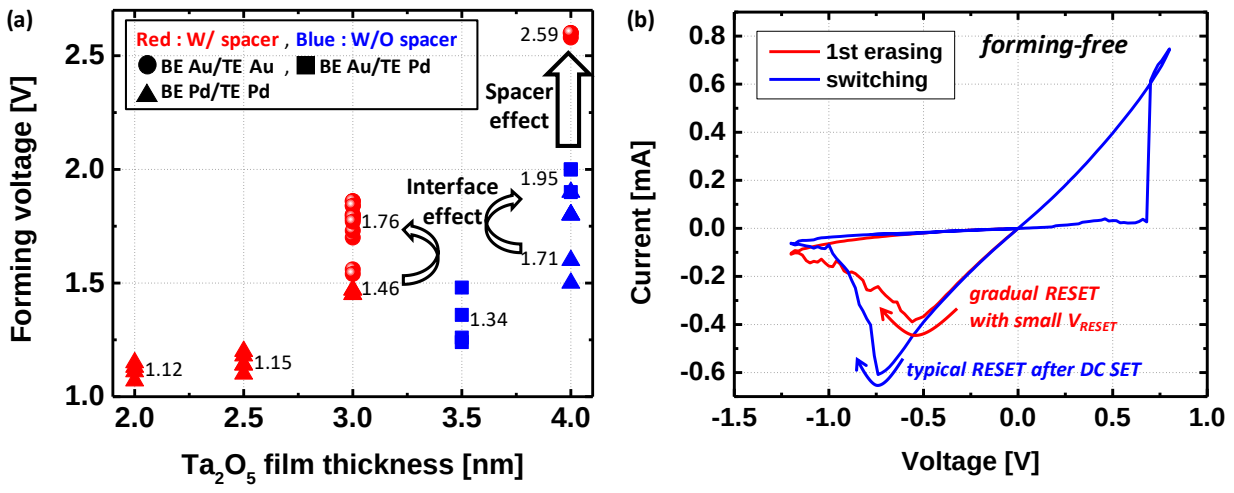

Figure 5-1: Forming-free device. (a) Trend of forming voltage as a function of the Ta_2O_5 layer thickness and device structure. (b) *I-V* curve of a truly forming-free device (initially 'ON' device). The first RESET curve shows more gradual reset with a small V_{RESET} compared with subsequent RESET process.

in digital form grows exponentially in area and power as the required bit lengths grows in larger arrays [82], [83]. Moreover, parasitic effects, such as line resistance and sneak-current issue, also become more severe and impede the operation of large arrays. Thus, our target size for practical demonstration of memristor network is ~ 32×32.

Among the challenges in the development of a larger array, high device yield is one of the most urgent issues that should be secured for stable implementation of neuromorphic algorithms. Specifically, in offline learning cases, fail (not working) cells in the memristor array significantly affect the accuracy of the network since information of the malfunctioning cells could not be taken into account in the training stage [87]. In Figure 2-3(c), we showed the yield of current optimized devices used for the array demonstration, where the device yield is defined by two types of yield: 'forming yield' which counts failure during the forming process and is mainly caused by process related intrinsic defect, and 'switching yield' which determines the yield during device switching after the forming process and is mainly driven by voltage stress in already-formed devices during

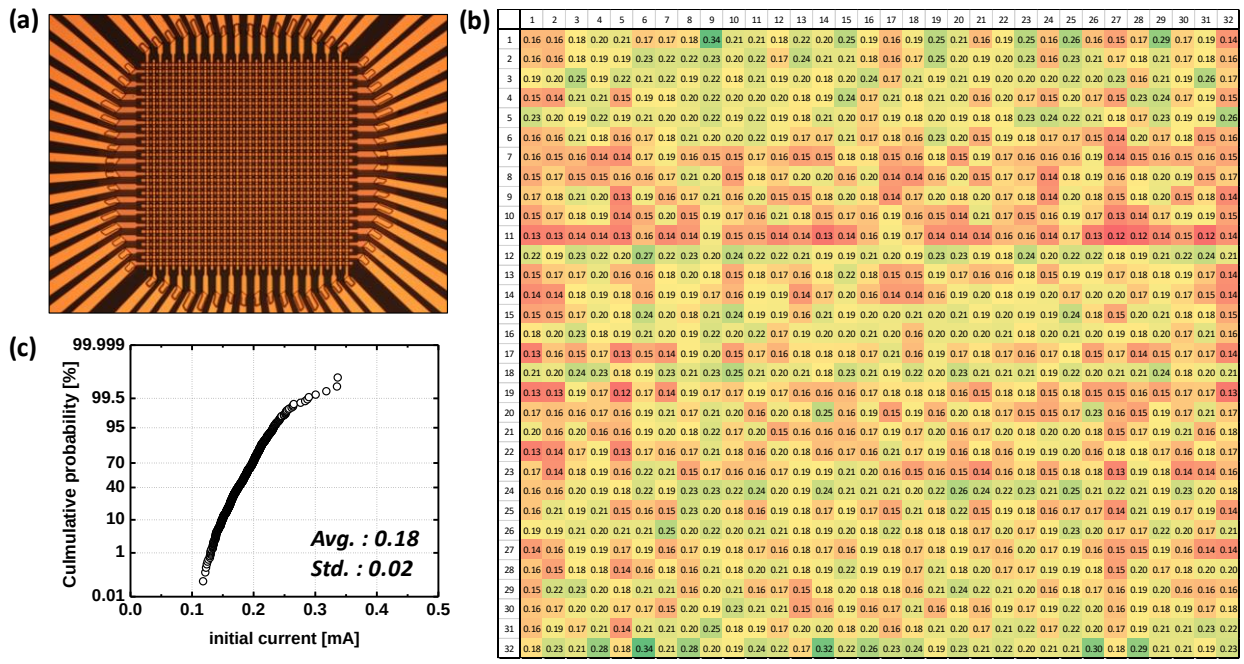

Figure 5-2: A 32×32 array of truly forming-free devices. (a) SEM image of the 32×32 memristor array. (b) Distribution of the initial read current of the 32×32 cells and (c) the cumulative probability curve obtained from (b).

forming or SET of other cells during the array operation. Thus, reducing the forming voltage (in some type of forming-free behavior) can significantly reduce voltage stress experienced by cells in the array and is strongly preferred to improve the device yield with stable switching.

Figure 5-1(a) plots the trend of the forming voltage, V_{form}, from experiments using different Ta_2O_5 film thickness and device structures. In an optimized structure the Ta_2O_5-based memristors can produce very low V_{form} that is close to V_{SET} around 1V, i.e., having < 2nm Ta_2O_5 with a spacer design and using Pd electrodes. We have also attained truly forming-free device (initially 'ON' state) by using extremely thin, i.e. 0.8nm Ta_2O_5 with optimized deposition power (30W), as shown in Figure 5-1(b), where the switching operation starts from RESET (red curve). As expected, almost 100 percent of yield was obtained from the forming-free memristor array, with uniform

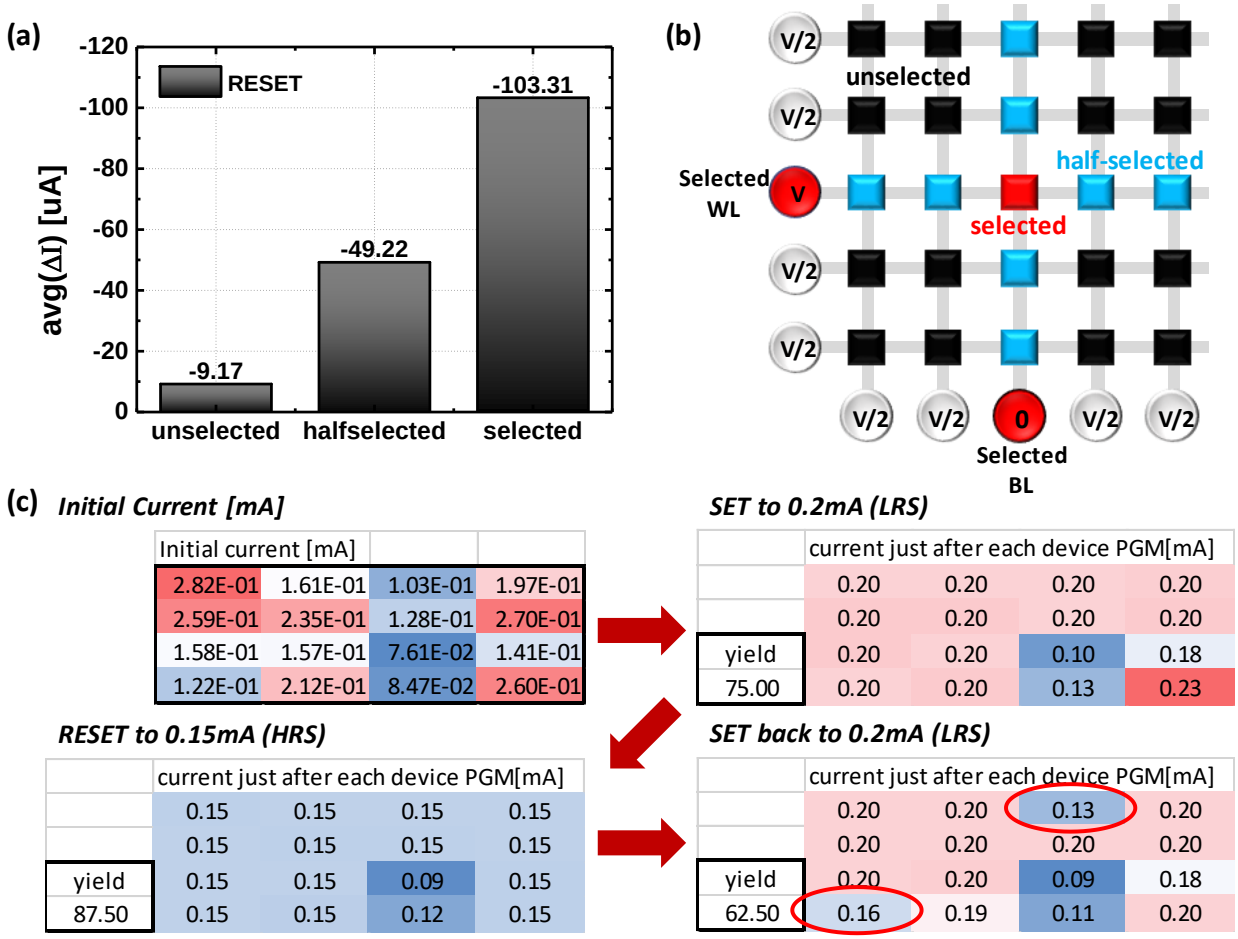

Figure 5-3: Potential issues in arrays based on truly forming-free devices. (a) Disturbance of half- and un-selected cells during RESET operation. Considerable disturbance appears in the half-selected cells due to lower RESET bias. (b) Schematic of the 1/2 protection scheme. (c) Degradation of the LRS. It becomes more difficult to SET as the device goes through cycling.

initial device current (at the 'ON' state) as shown in Figure 5-2. However, two issues still need to be addressed before arrays based on the truly forming-free type devices can be used.

First, the initially 'ON' devices are vulnerable to disturbance during the RESET operation. To program cells in an array, protective voltage methods, e.g. the 1/2 voltage scheme as shown in Figure 5-3(b), should be applied to prevent programming disturbance by reducing voltage across half-selected (sky blue) and unselected (black) cells. However, the initially formed devices may

not be as stable as conventional electrically-formed devices. Specifically, in the forming-free type, the conductive filament is formed during fabrication through an ion diffusion process rather than ion drift, and consequently the conductive filament is not as well defined and is more susceptible to be disturbed (i.e. easier to be RESET). As a result, during the RESET process of the target cells severe disturbance can occur in half-selected cells (Figure 5-3(a)). Strong SET operations such as using DC sweep or pulse trains with high voltages can create more robust filament and increase the V_{RESET} (Figure 5-2(b), blue curve), but they will necessitate an additional process that involves current-compliances to inhibit irreversible device break-down as addressed in the next section.

Second, the forming-free device may inherently hold less amount of *Vo*s due to the elimination of the forming step and can experience degradation of the LRS state during cycling, as shown in Figure 5-3(c). When repeating the SET and RESET processes with moderate pulse amplitude and duration in the initially 'ON' devices, several cells could not reach the target current values (0.2mA) during the SET operation and the number of failed devices grows with cycling. Again, DC sweeps or higher voltage pulses that can provide sufficient amount of *Vo*s can be used to address the SET degradation but require careful current compliance to prevent device break-down.

5.2 Integration with In-cell Resistor

From the previous section, even in truly forming-free (initially 'ON') devices, one-time SET operation with DC sweep or slightly higher pulse amplitude is beneficial to create stable conductive filaments with robust switching behaviors. However, positive feedback processes can occur during SET operation, reflected as the increase in E-field and temperature through Joule-heating as the filament grows which further speeds up the filament grow, can cause permanent device break-down that lead to devices stuck at the '1' state. Thus, we integrate each memristor

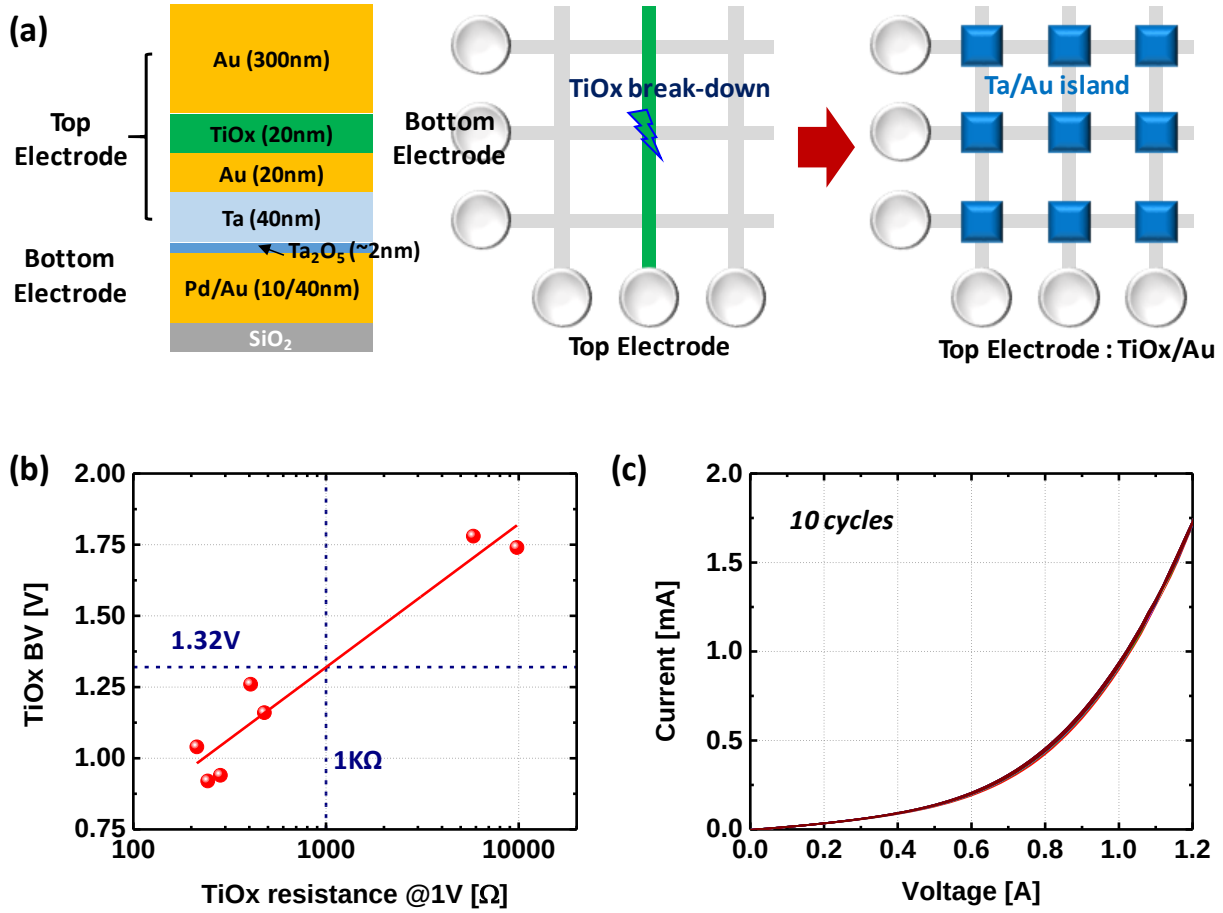

Figure 5-4: Integration of a TiOx-based in-cell resistor with a memristor. (a) Film stack and the proposed island-type structure of the in-cell resistor integrated with the memristor device. (b) TiOx break-down voltage (BV) as a function of the resistance value. At the target resistance of 1KΩ, TiOx BV is expected to be ~1.32V. (c) *I-V* curve of a TiOx test pattern with 10 cycles. No resistive switching in TiOx is observed up to 1.2V during voltage sweep.

device with an in-cell resistor which through the voltage divider effect reduces the E-field across the switching layer as the filament grows, thus providing a negative feedback that enables controlled filament growth. Given the device size (2μm×2μm), target resistance (1KΩ) of the in-cell resistor, and reasonable resistor film thickness for integration (e.g. 20nm), the resistivity of

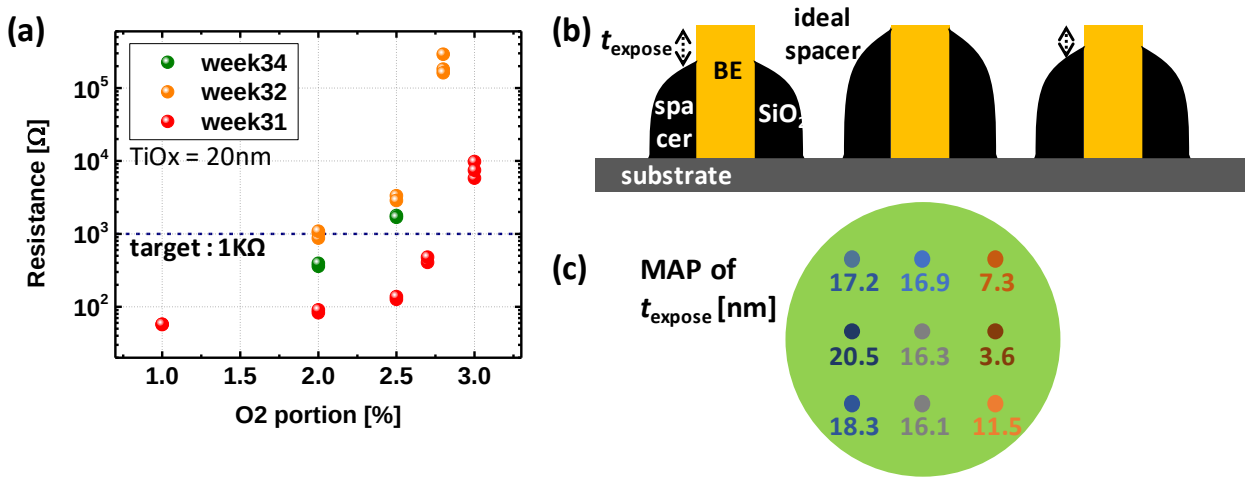

Figure 5-5: Process-induced variations. (a) Fluctuation of the TiOx resistance depending on week number (one batch per one week). (b) Schematic of spacer structure for the tall bottom electrode and (c) Variation of t_{expose} within a wafer. Every batch has up to 20nm of min-max differences.

The material used as the in-cell resistor should be in the region of 0.2Ω·m, which is still achievable in natural solid state materials.

We choose reactive sputtered TiOx to achieve such an intermediate resistivity value for the in-cell resistor. The film is placed on top of an inert Au layer that protects the Ta electrode layer to prevent interaction between TiOx and the Ta electrode or the Ta_2O_5 switching layer, as shown in Figure 5-4(a). The two parallel Au layers (along the column line), however, can bring a leakage current path for the cells sharing a column, once TiOx breaks down between the two layers. Thereby, an island structure with isolated Ta and the middle Au layer (Figure 5-4(a) right) is proposed to prevent cells from being shorted. The TiOx layer also needs to be carefully checked to verify the breakdown voltage (BV) and make sure resistive switching does not occur in the TiOx film itself. BV of TiOx at targeted 1KΩ is found to be about 1.32V in Figure 5-4(b), which is higher than the SET voltage required for stable switching of the Ta_2O_5 devices, and the film shows

identical *I-V* curves during 10 cycles of DC sweep without resistive switching behavior (Figure 5-4(c)), suggesting the TiOx film can be used as reliable in-cell resistors.

5.3 Process Variation within-Batch (= within-Wafer)

Up to now, we have looked into several geometries and structures to demonstrate a practical network sized 32×32. In actual device fabrication, numerous sources of device variations can exist, especially in a university level cleanroom. For example, evaporator tools to deposit the Au film varies around 10% in film thickness from batch-to-batch and more severe fluctuations occur during reactive sputtering deposition of films such as TiOx, as shown in Figure 5-5(a). Those variations make it difficult to obtain an ideal device. Far more serious is the variations within-batch (= within-wafer) than the batch-to-batch fluctuations. In our case, dry etching process to form the spacer structure causes the largest fluctuations within-wafer, as shown in Figure 5-5(b), and needs to be carefully optimized.

Additionally, large size of Ta_2O_5-based array with relatively high level of conductance (over than 1mS) requires thicker electrodes to reduce parasitic resistance. While thick top electrode (TE) deposited in the last step of fabrication process would not dominantly affect device fabrication procedures, the structural profile of the bottom electrode (BE) directly affects subsequent process steps, especially the critical Ta_2O_5 switching layer deposition step right after the BE formation. Thus, we built a spacer in Figure 5-5(b) to cover the sidewall of the thick BE, (typically ~ 300nm thick), through a sequential process of SiO_2 deposition (4μm) and reactive ion etching (RIE). However, this process results in detrimental within-batch variations in the height of the uncovered BE side-wall (t_{expose}), as shown in Figure 5-5(c), and causes column-wise different cell behaviors. To minimize device variations, the spacer structure can be eliminated and the BE thickness needs to be carefully designed through detailed HSPICE simulations to minimize series

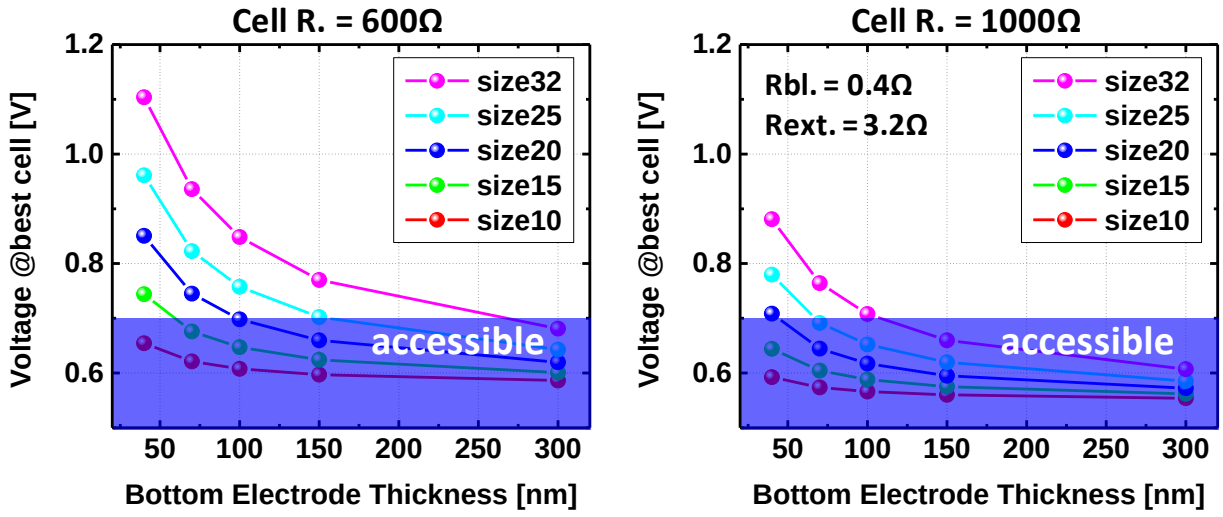

Figure 5-6: HSPICE simulation for two cases (LRS and HRS) to figure out the maximum array size as a function of the BE thickness. The blue box defines the tolerable regions. (left) cell resistance = 600Ω and (right) cell resistance = 1000Ω.

resistance effects during the array operation. The experimentally measured values of R_{column} (0.4Ω per column-wise segment) and $R_{external}$ (3.2Ω to reach the PAD electrode) are used in the simulation.

First, voltage drop at the best case (closest cell from both the row- and the column electrode, V_{best}) and the worst case (farthest cell, V_{worst}) is checked through the simulation after applying the same programing voltage, e.g. 1V, in both cases. Next, how much voltage needs to be applied to successively program the worst-case cell is obtained from the results, V_{worst}. Finally, we obtain the voltage drop across the best-case cell in the half-selected case, $(V_{best}/V_{worst})\cdot(1/2)$, when the worst-case cell needs to be programmed, for two cases of devices representing in LRS and HRS as shown in Figure 5-6, respectively.

Empirically, the Ta_2O_5-based memristor begins to be SET at 0.7V, therefore the safe range from programing disturbance is marked by a blue box representing voltage below this value. From

these results, we choose 40nm as the BE thickness which allows array size of 32×12 to safely operate using devices we already developed previously for *K*-means clustering and PDE studies. Figure 5-7 shows desired *I-V* curve of Ta_2O_5 memristor with 1KΩ in-cell resistor.

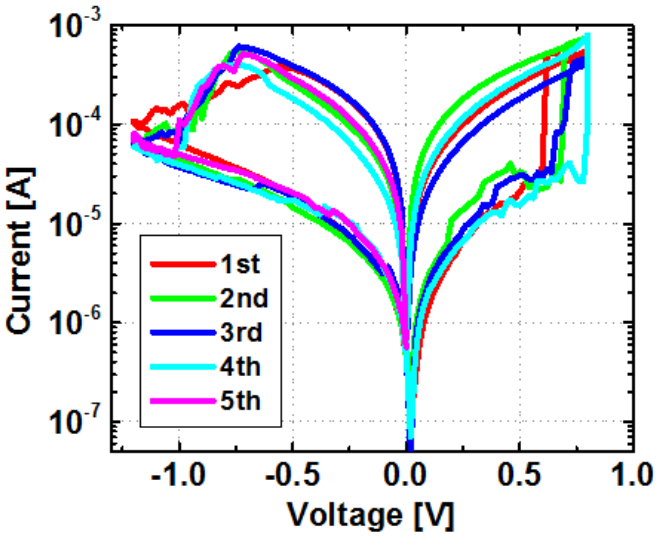

Figure 5-7: Desired *I-V* curve of a Ta_2O_5 memristor with 1KΩ in-cell resistor. The forming voltage is low enough to prevent cell damage and the C.C. effect from the in-cell resistor holds the current level below 1mA for stable switching.

5.4 Plasma Etching Damage

Finally, this section describes damage in devices during plasma etching of a large pattern. In Figure 5-2(c), the initial current level of normally forming-free cell should be in the range from 0.1mA to 0.3mA. Nonetheless, considerable portion of the as-fabricated array shows far wider distribution of the initial current, and moreover the majority of the cells cannot be successfully switched due to failure in the forming stage, as shown in Figure 5-8. Lastly, those arrays have quite different characteristics when compared with the stand-alone-cells fabricated in the same chip. Factors such as wire-bonding damage, plasma damage during deposition and etching, switching

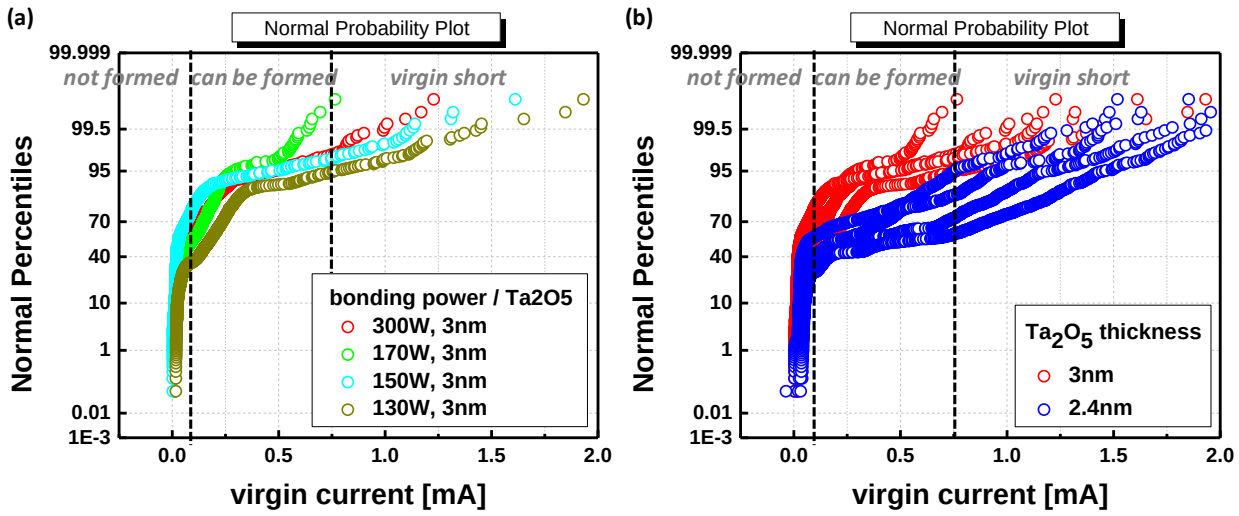

Figure 5-8: Normal probability plot of the initial current for cells in the array. (a) Bonding power does not affect the current distribution. (b) Devices with thinner Ta_2O_5 thickness are vulnerable to be damaged during the RIE etching process.

layer thickness variation, etc., were investigated. The cell damage in the array was eventually traced to the damage of Ta_2O_5 switching layer during reactive ion etching process to open the BE connection. In particular, devices with thinner switching layer are more prone to plasma damage and result in wider distribution, as shown in Figure 5-8(b). The distribution tightens with reduction of the plasma etch time. Additionally, stand-alone cells fabricated on the same chip have small metal patterns that need to be etched, and are thus less affected by plasma damage. In order to solve this issue, all masks were re-designed to have smaller metal patterns on which high energy of plasma charging can occur, especially during the RIE etching process.

5.5 Conclusion

This chapter introduced several guidelines for the fabrication of practical sized memristor arrays. To obtain stable switching behaviors, a forming step is beneficial even if the applied V_{form}

is not prominently higher than the SET voltage. Unlike individual devices, cells in the array are more susceptible to process-induced damages as well variations at each process step, thus more careful design of the device structure and fabrication procedures are required.

Chapter 6. Parasitic Effects Analysis in a Memristor Network

In Chapter 5, guidelines regarding the fabrication of practical sized memristor arrays were introduced through experimental and simulation results. From the network operation point of view, parasitic effects, such as the sneak path leakage and the parasitic resistance, can still pose significant challenges in a network formed by purely passive memristor devices [38], [88], [89] and thus requires careful analysis, which is the topic of discussion of this chapter.

6.1 Array Operation Conditions and Parasitic Effects

In a memristor-based neuromorphic system, VMM is naturally obtained through the network using Ohm's law and Kirchhoff's current law as mentioned in the previous chapters. We note that the operation conditions of the VMM (or referred as dot product), can be considerably different from

operation		RRAM		neuromorphic
		read	write	dot product
bias	sel. WL/BL	V_{read} / GND	V_{write} / GND	All WLs : V_{input} All BLs : GND
	unsel. WL/BL	F or GND	F or $V/2$ or $V/3$	
direction		forward		forward, backward
array pattern		0 or 1		intermediate states

Figure 6-1: Different operation conditions in memory and neuromorphic applications. Unlike during RRAM memory operations where most devices are on unselected WLs and BLs, during dot product operations all the WLs and BLs are connected to known potential values. As a result, the sneak path effect is reduced, while the series resistance problem may be amplified.

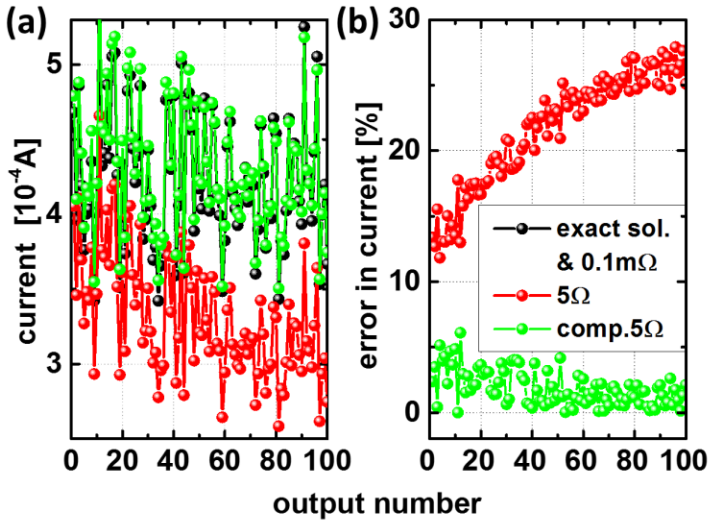

Figure 6-2: Current distortion by R_{line}. (a) Output currents and (b) the relative error obtained at each neuron during dot product operation in a 100×100 array. The memristor weights and input patterns are randomized. In the small R_{line} case, 0.1mΩ, the results exactly matched with the exact solutions (i.e., results of the numerically calculated dot product), indicating negligible series-resistance effects during dot product operation if R_{line} is small enough. As R_{line} increases (5Ω case), the output currents decrease, with large numbered neurons (the ones further away from the inputs) experiencing larger distortions. The proposed scaling approach provides remarkable compensation capability (plotted as comp. 5Ω), where the error is significantly reduced and consequently the non-uniform degradation of the output currents is suppressed.

the read operation in RRAM memory applications, while the write operation conditions are similar, as listed in Figure 6-1. Specifically, during the dot product operation, all WLs (row-side) are activated and all BLs (column-side) are connected to sense amplifiers through virtual ground, while during a memory read operation, only limited number (e.g. one) of WL and BL are selected simultaneously and the unselected WLs and BLs are either left floating or biased at different voltage levels, depending on the scheme of the operation [90]. As a result, the BL potentials are normally well-defined during the dot product operation, which can help mitigate the sneak-current

path problem. However, the accumulation of currents through all the activated WLs and BLs can lead to pronounced series-resistance problem, as will be made clear below.

Figure 6-2 shows an example of the parasitic effects during dot product operation by HSPICE simulation in a 100×100 memristor array. The weight and input pattern are random and device resistance is ranging from 30K to 300K. It is notable that the extremely small R_{line} case (0.1mΩ) exactly match with the numerical dot product solution (the exact solution, black), implying negligible series-resistance effects, as expected. More importantly, as the parasitic resistance increases, the output deviates from the expected exact solution (red line) due to the line-resistance effect and the output current decrease is non-uniform, i.e., neurons with larger numbers (further from the input) will suffer more since they will experience stronger series resistance effects. This effect can also be clearly observed by plotting the relative error as a function of the neuron location (Figure 6-2(b)), where the error is calculated by comparing the current collected at the column with the exact, numerically calculated dot-product. The non-uniform distortion of the output current can lead to incorrect neuron firing or pattern mismatching, and consequently will limit the size of a practical memristor array, which can significantly affect the system's performance since larger arrays are desirable by allowing a larger dictionary size and offering higher level of parallelism.

We perform systematic simulations on 4 parameters; On/Off ratio, line-resistance (R_{line}), array size, and weight pattern (for (a) random and (b) trained by the popular Softmax function often used in machine-learning algorithms) as shown in Figure 6-3. From the reported articles, device On/Off ratio varies from 2 to 100 for analog memristor operation [36], [51], [91]–[98] and the line resistance R_{line} can be up to 10Ω for nanoscale devices [90], [99]–[104]. In the simulation,

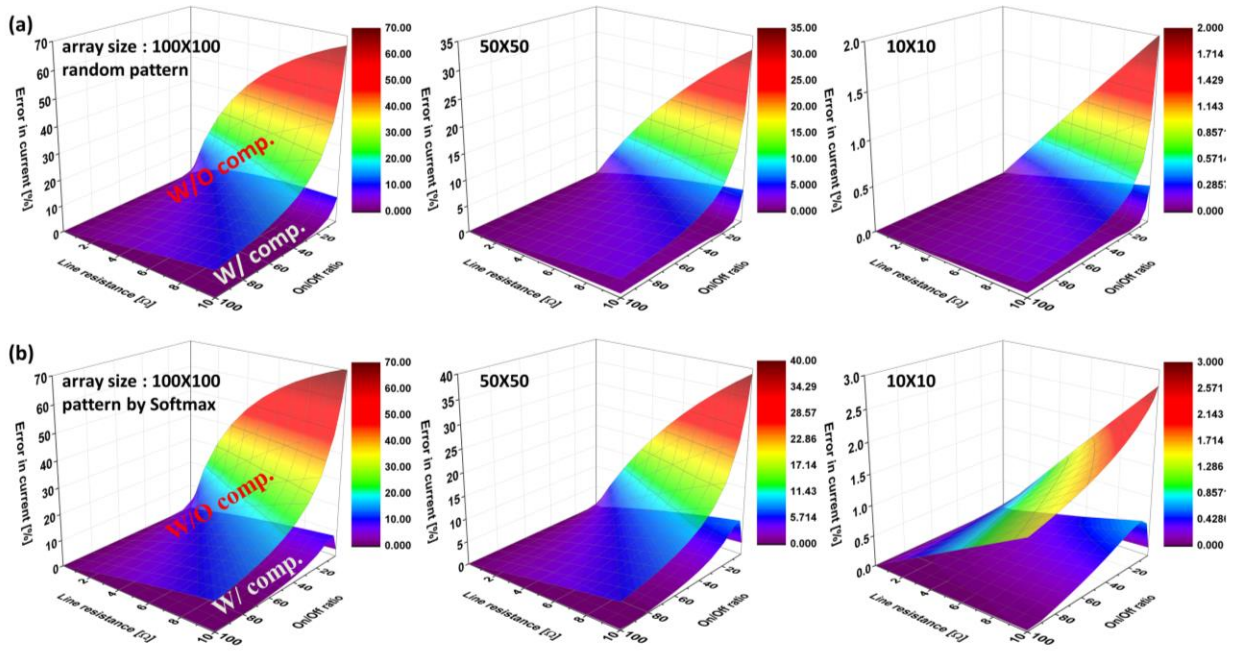

Figure 6-3: Systematic simulation for different R_{line} (0.1mΩ-10Ω), On/Off ratio (2-100), array size, and weight patterns ((a) random and (b) Softmax), showing the relative error from the raw data (W/O comp.) and after the simple scaling approach (W/ comp.). Random input patterns were used in the simulation. The parasitic resistance effects become worse as R_{line} and array size increase, and On/Off ratio decreases. The proposed simple scaling approach provides excellent compensation capability in all cases.

randomized input patterns are applied and we show the resulting error of the last column, since it will experience stronger series resistance effects (worst case). As expected, in all the cases, the parasitic resistance effects become worse as R_{line} and array size becomes larger, and On/Off ratio becomes smaller (the W/O comp. plots in Figure 6-3).

Reducing the series resistance or increasing the internal memristor resistance will help alleviate this problem, but these approaches are limited by the available material choices and practical device geometry. In addition, device nonlinearity, which can alleviate sneak path issues for memory applications by suppressing currents through unselected cells, is not helpful for the

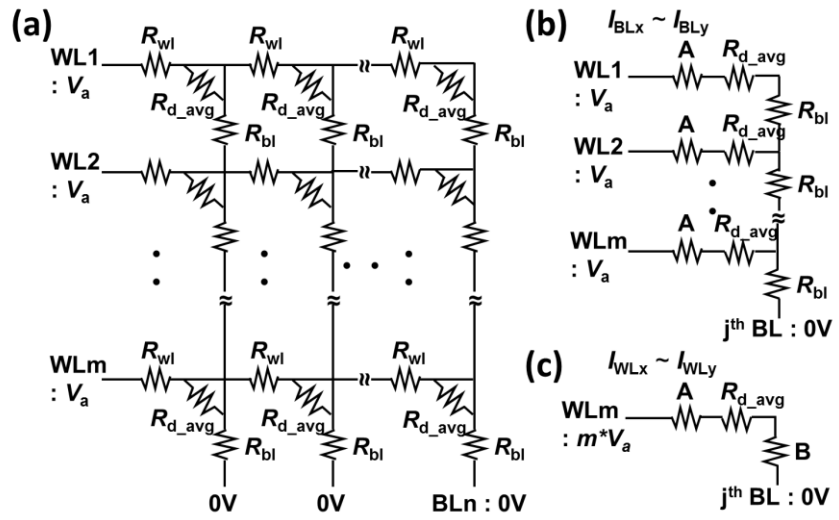

Figure 6-4: An equivalent circuit of the m×n memristor array. (a) Line resistances between cells are R_{wl} and R_{bl}. For simplification, each memristor is assumed to have the same average conductance, $G_{avg}=1/R_{d\_avg}$. (b) If R_{wl} is small compared with R_{d_avg}, we can assume the collected current at each output, I_1, is roughly identical. As a result, the current flowing through the j^{th} R_{wl} is $(n+1-j)\times I_1$, and the current flowing through the series-resistance causes a potential drop in the WL, which is modeled with a resistor with resistance A in Eq. (6-1). (c) Extending the idea into WL direction. From the modeled equation of I_{ideal} and I_{appr}, an extremely simple and practical relationship is derived.

problem discussed here since during the VMM operation all cells in the matrix are selected. In fact, having non-linear devices may worsen the problem, since the lower voltage-drop in the columns that are far from the input will lead to further reduced current in a non-linear device and exacerbate the current distortion effect.

6.2 Modeling and Solution

Here we propose a system-level solution that enhances the robustness and reliability of the neuromorphic systems even in the presence of the device limitations. As a first step, we examine the memristor crossbar circuit in Figure 6-4(a) and simplify it to an equivalent circuit model for

ease of analysis. The memristor array is an m×n matrix, and we further assume all inputs are fixed at level V_a and all devices have the same average conductance (R_{d_avg}). The reason is that our goal here is to model the complex resistive network as simply as possible without losing the essential properties that can affect the network operation. We will show that the simplified model can successfully capture the output distortion effects.

Since all the BLs are connected to virtual ground and the word line resistance R_{wl} is small compared with the average memristor resistance R_{d_avg}, the current flowing through each memristor on the same WL is roughly identical, i.e., the currents through the x^{th} and the y^{th} memristor on WL i are roughly the same, $I_1=I_{ix}=I_{iy}$. As a result, the current passing through each segment R_{wl} in WL i will be an integer multiple of I_1, determined by how many devices' current flows through this segment. A simple analysis shows that the current flows through the j^{th} R_{wl} is $(n+1-j) \times I_1$, i.e., cells that are located after this segment contribute to the current, while cells that are located before the segment do not (since their currents flow directly to the respective BLs).

Based on this analysis, we can greatly approximate the circuit. Specifically, the series-resistance due to R_{wl}, combined with current through it, reduces the actual potential that is applied to a memristor cell. The potential distortion in turn causes current distortion in the end and can be modeled using a simplified circuit in Figure 6-4(b). Here all R_{wl} segments on a given WL can be modeled with a resistor having resistance A

$$A = \sum_{k=1}^{j}(n+1-k) \cdot R_{wl} \quad \text{(for } j^{th} \text{ BL)} \quad (6\text{-}1)$$

where the effect of each R_{wl} segment depends on its distance from the input, as discussed above. The R_{wl} segments after BL j does not decrease the potential on BL j and their effects are not considered.

Extending this argument along the BL direction, we obtained a simplified equivalent circuit as shown in Figure 6-4(c), with

$$B = \sum_{k=1}^{m} k \cdot R_{bl} \quad \text{(for } m^{\text{th}} \text{ WL)} \qquad (6\text{-}2)$$

Eq. (6-3) shows the approximated output current, I_{appr}, based on this simplified circuit. When compared with the ideal current output that does not consider series-resistance effect, $I_{ideal}=mV_a/R_{d\_avg}$, the model leads to a remarkably simplified but useful expression as shown in Eq. (6-4), where the V_a term can be canceled out.

$$I_{appr} = m \cdot \frac{V_a}{A+R_{d\_avg}+B} \qquad (6\text{-}3)$$

$$I_{ideal} = I_{appr} \cdot (A + R_{d\_avg} + B)/R_{d\_avg} \quad \text{(for } j^{\text{th}} \text{ BL)} \qquad (6\text{-}4)$$

Next, we discuss how to choose R_{d_avg} values. Specifically, R_{d_avg} needs to be a value between minimum (R_{min}) and maximum (R_{max}) of the device resistance. For the proposed simple model, it should also be obtained from known parameters. Here, we applied weighted average of R_{min} and R_{max} as shown in Eq. (6-5). The weight a (for R_{min}) and b (for R_{max}) are set to decay exponentially as the device resistance value increases (Eq. (6-6)). This rule is based on the intuition that the impact of parasitic resistance on the network is determined by the relative values of the device resistance and the parasitic resistance and thus the parasitic effect will become smaller as the device resistance increases.

$$R_{d\_avg} = \frac{(a \cdot R_{min} + b \cdot R_{max})}{(a+b)} \qquad (6\text{-}5)$$

$$a = (R_{min})^{-k} \ , \ \ b = (R_{max})^{-k} \qquad (6\text{-}6)$$

, where k is chosen empirically from the simulation that leads to the best results from Eq. (6-4), and 0.17 is used for randomized weight patterns. For weight patterns trained using a specific leaning rule, k depends on the learning rule used in the network and array sizes (e.g., for Softmax

pattern, k=0.23, 0.42, 0.90 (for array size 100, 50, and 10)). Since the learning rule and the array size are known parameters, k only needs to be decided once for each case.

Finally, from above equations, the current ratio, I_{ideal}/I_{appr} in Eq. (6-4), can be obtained at each output neuron j and provides valuable information such as scaling factors or threshold values that need to be adjusted to account for the parasitic effects. Specifically, analysis of the current ratio allows one to compensate the reduced current values and recover (nearly) ideal values at the respective output nodes, and consequently allow the network to function properly in the presence of these non-idealities. Moreover, it should be noted that the approach only involves few common parameters that can be known beforehand, such as the array size, the nominal series resistance at each segment, and the output neuron number, independent of the input and the weight patterns.

6.3 Compensation of the Parasitic R_{line} Effect

Even with the simplicity of the proposed approach, it displays excellent capability to removes the current distortion in the network and produces results close to the ideal cases. As shown in Figure 6-2, without compensation, larger number neurons (BLs that are further from the input) will receive progressively lower output currents (red dots) due to the more severe series-resistance effects these neurons experience. After scaling the output using Eq. (6-4), the output currents (green dots) essentially recover to the ideal case (black dots). The error, measured as the difference from the exact solution, also shows notable decrease and remarkable flatness after compensation. The proposed simple model can compensate the distorted current outputs and suppress the error considerably for practical ranges of R_{line}, On/Off ratio, and array size as shown in Figure 6-3 (marked as W/ comp. plots). To further test the applicability of the proposed method, we studied different types of weight patterns trained by different learning rules, e.g., Softmax and Oja's rule in Figure 6-3(b) (Oja's rule case will be shown in the next sub-section) and the proposed model

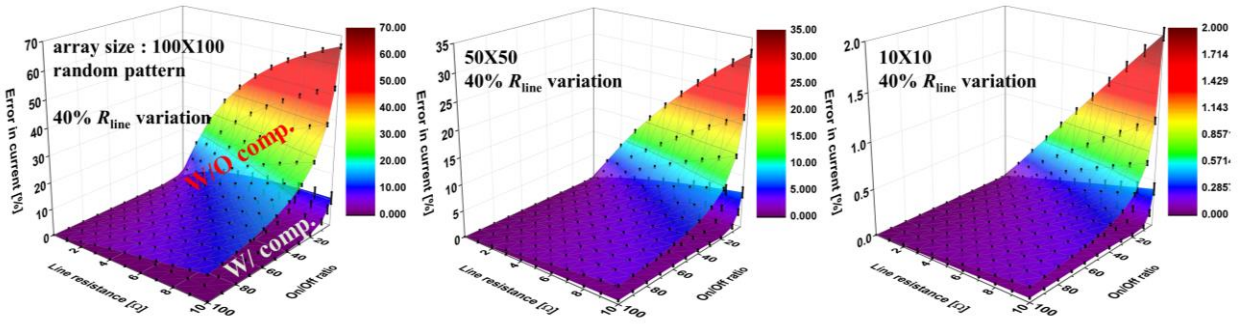

Figure 6-5: HSPICE simulation results considering R_{line} variations at each wire segment, for an extreme case with 40% R_{line} variations. Random weight and input patterns are used for the analysis. The black error bars represent the standard deviation obtained from 10 repeated simulations. Even with severe R_{line} variations, the impact on the output current is minimal and the proposed approach is able to compensate the error effectively using only the nominal R_{line} value.

can effectively address the series resistance problem for different weight patterns trained by different learning rules as well.

In a practical implementation, R_{line} (R_{bl} and R_{wl}) at each parasitic resistance segment can vary significantly after array fabrication, while we utilize fixed nominal value in the modeling. To check the impact of R_{line} variation on the proposed model, we reproduced the systematic simulation in Figure 6-3(a) by introducing R_{line} fluctuation. We assume R_{line} at each segment exhibit a Gaussian distribution around the nominal value and considered extreme conditions (40% standard deviations). Even with these severe R_{line} variations, the output current maintained almost the same error for all cases in 10 repeated simulations (error range represented by the black-bars at each point in Figure 6-5). More importantly, the proposed method still provides good compensation ability using only the nominal R_{line} values. This small impact of R_{line} variation can be understood from the averaging effect of the large number of R_{line} segments, ranging from 200 (for size 10×10) to 200,000 (for size 100×100).

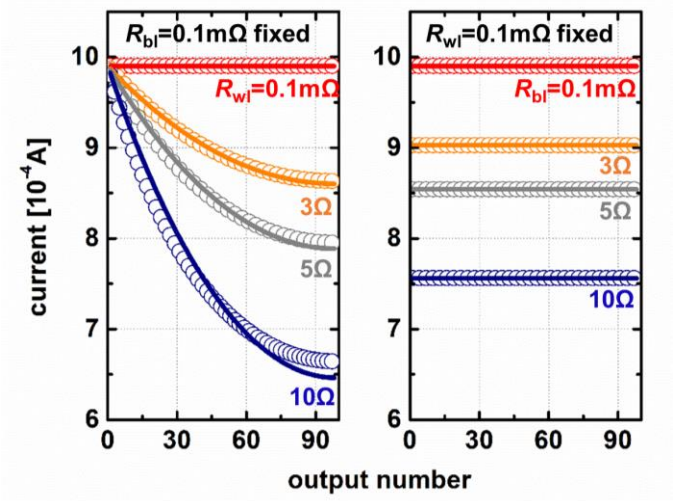

Figure 6-6: Comparison of the effects from WL and BL series resistance, R_{wl} and R_{bl}. Open symbols represent results from the proposed simple model in Eq. (6-4) and solid lines represent results obtained from HSPICE simulations. The weight pattern is fixed to all 100KΩ for the simulations, and the input pattern is 1V for all WLs. It can be observed that for forward dot product operation, R_{wl} causes non-uniform distortion (left), while R_{bl} only uniformly reduces the output current (right). This implies that if a neuromorphic algorithm only involves forward direction dot product operations, R_{wl} poses a much more severe problem than R_{bl}.

Another question the model helped address is which series resistance, R_{wl} or R_{bl}, causes more severe problems. We simulated the effect of R_{wl} and R_{bl} separately by using two methods, the model presented in Eq. (6-4) (opened symbol) and more accurate HSPICE (solid line). The weight pattern is fixed to all 100KΩ and the input pattern is 1V for all WLs. As shown in Figure 6-6, the simple model excellently captures the results of the HSPICE simulation, once again confirm the validity of the proposed model. More importantly, during forward dot product operation, only R_{wl} causes nonuniform current degradation along the BL outputs, while R_{bl} only uniformly decreases the output current and will not affect the identification of the winning neurons. Therefore, if the algorithm only involves forward dot product operations, e.g. in a purely

feedforward network, R_{bl} will not significantly degrade the system performance even though it may result in more iterations to cause neuron firing. Consequently, extending output neuron number n (which is affected by R_{wl}) is more challenging than expanding the input size m (which is affected by R_{bl}). The opposite is true if the network operation also requires backward dot product operations, such as in sparse-coding algorithms [37], [59].

The presented approach relies on scaling the crossbar output current to compensate the parasitic line resistance effect. The scaling factors are identical for all systems having the same device properties, network size and learning rules. The scaling factors also remain constant during the operation of the network and hence can be directly realized at the post-neuron circuit. For instance, the interface capacitors of the post-neurons can be fabricated to have the scaled sizes that reflect the line resistance compensation coefficients obtained beforehand. In this case, little extra cost is introduced to the hardware. If different neural algorithms need to be run on the same hardware, a more flexible approach can be implemented at the neuron ADC level by scaling the ADC's reference voltage [105]. This approach is also easily implementable at minimal extra cost.

6.4 Parasitic Effects in a Sparse Coding Algorithm

In this section, we analyze the parasitic effects during a complete neuromorphic operation, using a feature extraction algorithm that is believed to underlie higher level cognitive functions [37], [66], [106]–[109]. The feature extraction operation consists of two steps: first, learning of the receptive fields (dictionary elements), achieved by updating the memristor devices using a training algorithm; and second, feature detection, which aims to find an optimal representation of the input while minimizing a cost function.

We implement training of the dictionaries using Oja's rule in Eq. (6) [110], coupled with a winner-take-all (WTA) strategy [111]. This approach allows fast training and is compatible with

the memristive hardware, where the winners can be readily identified from the forward dot product operation in a memristor crossbar structure. After identifying the winning neuron from the largest dot product, the corresponding synaptic weights are updated according to Eq. (6-7).

$$\Delta\Phi = \beta(X - y\Phi^T)y \ , \ y = X \cdot \Phi \quad (6\text{-}7)$$

where, β is the learning rate, X is the input vector, Φ is the matrix of receptive fields and represented by the memristor conductance values.

After dictionary learning, the locally competitive algorithm (LCA) [112], [113] was used to perform feature detection. Detailed description of the LCA can be found in [66], [112] and a brief introduction is provided here.

$$\frac{du}{dt} = \frac{1}{\tau}(-u + (X - a \cdot \Phi^T) \cdot \Phi + a) \quad (6\text{-}8)$$

$$a = T(u,\lambda) = \begin{cases} u, & if\ |u| \geq \lambda \\ 4u - 3\lambda, & if\ 0.75\lambda \leq |u| \leq \lambda \\ 0, & if\ |u| \leq 0.75\lambda \end{cases} \quad (6\text{-}9)$$

where, u is the neuron's membrane potential, X is the input vector, Φ is the matrix of receptive fields, and a represents the activities of the neurons determined by a thresholding function $T(u,\lambda)$ with threshold λ. During LCA, the membrane potential of the output neurons, u in Eq. (6-8), is determined as following: the dot product $X \cdot \Phi$ term reflects the similarity between the input and the dictionary element, while the term $-a \cdot \Phi^T\Phi$ inhibits neurons with similar receptive fields from firing together and help satisfy the sparsity requirement. The neuron membrane potential also spontaneously decays due to a leakage term $-u$ with time constant τ. The membrane potential is then compared with a pre-defined threshold, λ, to convert to the activity of the neuron, a, as shown in Eq. (6-9). Experimentally, the inhibition term $-a \cdot \Phi^T\Phi$ can be obtained by performing a backward pass to obtain a reconstructed input $a \cdot \Phi^T$, followed by forward pass through the same

memristor network. After repeating these cycles and reaching stabilization, the algorithm yields a small number of active neurons and the original input can be reconstructed using the sparse code *a* and the associated receptive fields. It should be noted that forward and backward dot product operations are needed during both the training and the LCA feature detection stages, as shown in Eqs. (6-7)-(6-9).

6.4.1 Training of Dictionaries

The parasitic effect during training was analyzed using HSPICE simulation during both forward and backward dot product operations. Since training is a sequential process, calling HSPICE at every dot product operation is very time-consuming [66]. Thus, 10000 image patches, which is enough for examining the impact of series resistance were used for analyzing training effects, instead of possibly millions of training patches to obtain a fully optimized dictionary. The image patches are extracted from 10 different natural images and training is performed in a 49×98 memristor array. Even though the sneak currents are negligible in a neuromorphic operation as discussed in the previous section, the R_{line} effect significantly degrades the output current along the output neuron numbers. Therefore, neurons close to the input side in general win much more frequently during the forward dot product operation as shown in Fig. 6-7(a). As a result, only a limited number of dictionary elements that are associated with small numbered neurons receive training, and the rest of the dictionary remains at their initial random weights. This can have significant consequence of the system's performance since not only does it produce useless receptive fields for the subsequent analysis process, but it also eliminates the inherent self-adaptation capability of the neuromorphic system during online learning. In addition, series resistance effects during the backward dot product operation causes another problem during training. According to Eq. (6-8), the weight update is proportional to the error between the original

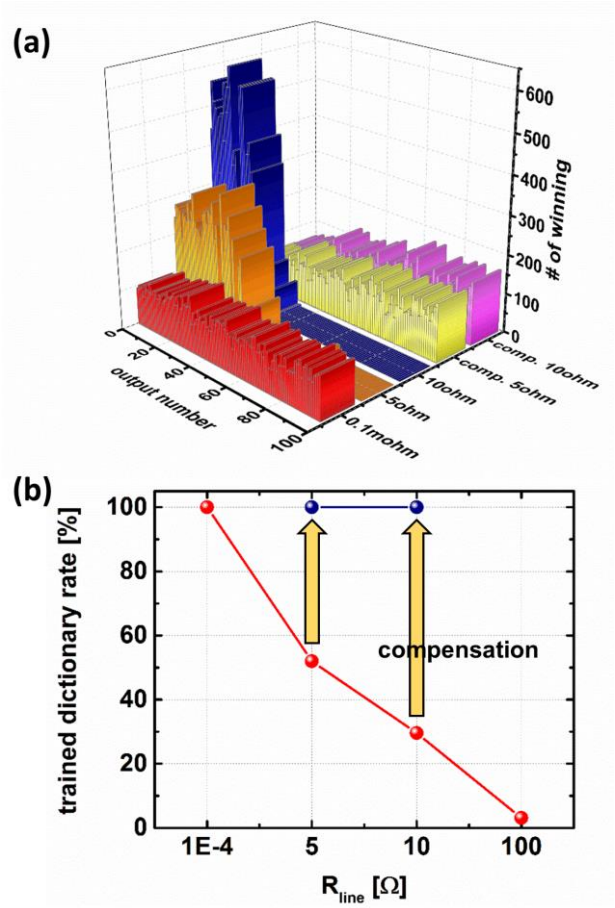

Figure 6-7: Parasitic effects during the learning stage. 10000 patches extracted from 10 different natural images and a 49×98 memristor array are used in the analysis. (a) Although the sneak path is negligible during neuromorphic operations, R_{line} considerably reduces the output current for neurons far from the input. Consequently, only a small number of neurons will repeatedly win and become trained, while others do not receive training at all. The proposed scaling method effectively compensates this effect, and uniform training is restored in (b).

image and the reconstructed image obtained from the backward process. Given the current reduction along the BL by R_{line} in the backward dot product operation, memristors far from the output side would always show a more positive Δw than the ones near the output side and this leads another error factor during learning. As a result, the series resistance effects cause learning

to be severely degraded, both in terms of the number of trained dictionary elements as well as in terms of the quality of the dictionary elements that do get trained.

We note the series resistance effect essentially sets a practical size of the dictionary that can be implemented, regardless of the physical size of the memristor array since dictionary elements beyond a certain number will not be sufficiently trained. To address these problems, the scaling approach developed in the previous section was applied in the simulation and results after the simple compensation scheme based on Eq. (6-4) analyzed, as shown in Figure 6-7 for R_{line}= 5Ω and R_{line} = 10Ω cases (labeled as comp. 5Ω and comp. 10Ω). From Figure 6-7(b), it can be seen that with 5Ω of R_{line}, without compensation the number of neurons trained at least once during the 10000 training patches is ~ 50% of the total neurons, and the degradation of training gets worse with higher R_{line}. The simple compensation model, however, effectively mitigates the series resistance effect and allows all neurons to be effectively trained.

6.4.2 Parasitic Effects during Feature Detection

The parasitic effects are then analyzed through simulations during the image analysis stage using the LCA algorithm with ideally trained dictionary elements to examine the effects independently from the training step. The dictionary is a collection of vectors representing feature primitives (also called receptive fields). For example, in image analysis, during learning the dictionary elements evolve to resemble feature primitives such as Gabor filters of different orientation and frequency, as shown in Figure 6-8(b). These feature primitives are thought to exist in the mammalian visual cortex [114] and are useful for identifying edges and textures [115], [116]. In the LCA algorithm, the input images can be encoded with a set of such dictionary elements using the activity of the associated neurons. A natural image in Figure 6-8(a) is used in the analysis and results for different sparsity factors (determined by the threshold λ are analyzed, as shown in

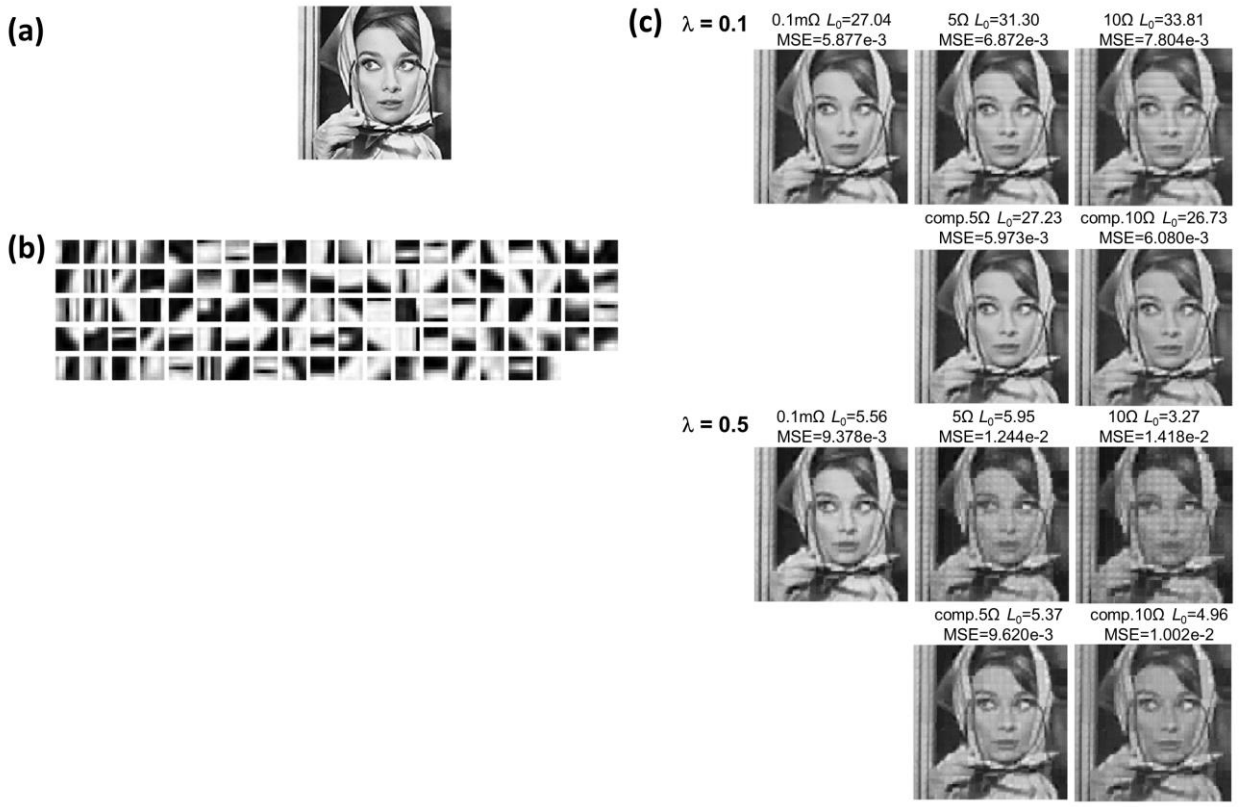

Figure 6-8: Original and reconstructed images. (a) The original test image. 7×7 non-overlapping patches were analyzed using LCA to reconstruct the original 140x140 resolution image. (b) Dictionaries obtained after training process. 98 dictionaries are shown here. (c) Reconstructed images obtained from LCA using the backward dot product and coefficients *a*. Larger series resistance leads to degradation of the reconstructed image.

Figure 6-8(c) [66]. To quantitatively measure the degree of degradation caused by the parasitic effect, the mean squared error (MSE) between the original image pixels and the reconstruction is calculated and summarized in Figure 6-9(a). First of all, we note for the same line resistance the MSE decreases as the L_0 norm increases. This is intrinsic to the sparse coding algorithm. Specifically, the image is reconstructed by linearly summing the receptive field (dictionary) elements, weighted by the activated membrane potentials (activities). Increasing the number of

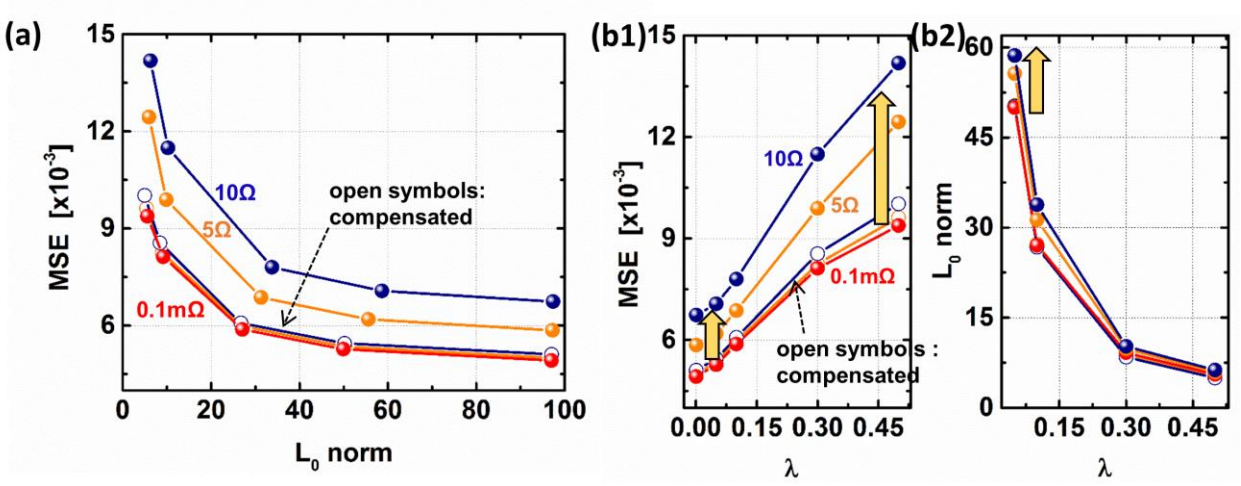

Figure 6-9: Effects on feature detection using LCA with the parasitic resistance problem. (a) 5Ω R_{line} increases the MSE by ~20%, and 10Ω R_{line} further worsens it to 40% for the same sparsity. The compensation method effectively restores the data close to the MSE of the 0.1mΩ case. (b) Dependence of the MSE and L_0 on the threshold parameter λ. The MSE degrades more at larger λ, while L_0 norm changes more at smaller λ.

active neurons (i.e. a larger L_0 norm) allows the network to utilize more features from the dictionary and consequently will in general improve the quality of the reconstruction, at a cost of reduced sparsity. Next, with the presence of R_{line}, the MSE increases by ~ 20% in 5Ω compared with the 0.1mΩ case, and 10Ω R_{line} further worsens MSE to 40% for the same sparsity factor.

The simple scaling approach can again effectively compensate for this error and results obtained using the compensation method produce similar MSE as the (negligible) 0.1mΩ R_{line} case, as shown in Figure 6-9(a). Figure 6-9(b) replots the results against the threshold λ, which is the parameter that is practically tunable in a real application. The MSE is more prone to be degraded by R_{line} at higher λ, while the sparsity factor (represented by L_0 norm of the active neurons) degrades more at lower λ. This can be explained by Eq. (6-10) [112], which represents the energy function the LCA algorithm aims to minimize:

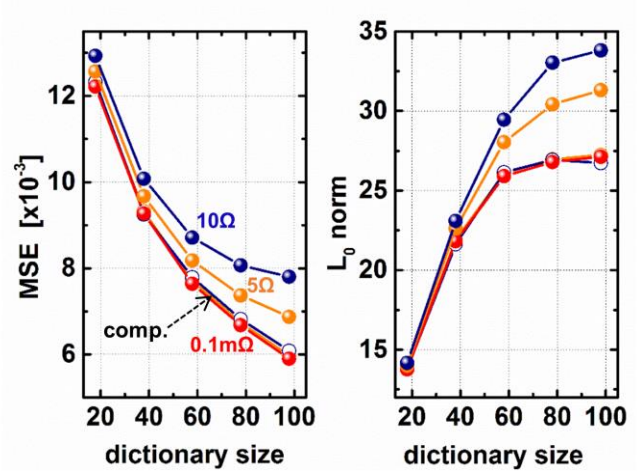

Figure 6-10: The parasitic effect for different dictionary sizes, with fixed λ, 0.1. (left) Larger dictionary size is generally helpful for feature extraction and reduces the MSE. However, the benefit is largely negated with high value of line resistance due to the parasitic effect. (right) With increased line resistance, the sparsity is also degraded.

$$\min_{a,\Phi}(\|X - a\Phi^T\|_2^2 + \lambda\|a\|_0) \qquad (6\text{-}10)$$

where $\|\cdot\|_2$ is the L_2 or Euclidean norm, and $\|\cdot\|_0$ is the L_0 norm, which is simply a count of non-zero elements. Thus, the LCA aims to find the stable minimum point in a 2D energy map that includes both the reconstruction error and the sparsity effects. When the array is affected by the series resistance, the obtained error term from the hardware system (1st term in Eq. (6-10)) will increase and the original solution (without the parasitic resistance effect) no longer represents the minimum total energy point. The system then tries to search for a new minimum point by adjusting both the error term (1st term in Eq. (6-10)) and the sparsity term (2nd term in Eq. (6-10)). The reconstruction error term (MSE) can be reduced by increasing the L_0 norm (e.g., using more active neurons). However, if the threshold λ is large, the sparsity penalty (2nd term in Eq. (6-10)) associated with increasing L_0 becomes higher, causing the total energy to increase and thus limiting the system's ability to minimize MSE in the presence of these non-idealities. As a result, the MSE

for systems with large R_{line} will degrade more at larger λ, while at smaller λ a larger increase of L_0 norm can be tolerated. These results are can be clearly observed in Figure 6-9(b1) and 6-9(b2).

Figure 6-10 represents the impact of the dictionary size on the LCA performance under a fixed λ value of 0.1. Larger dictionary leads to better feature detection, but increase of the line resistance significantly negates the array size advantage. By applying the simple compensation approach, the series-resistance effect can be greatly mitigated in all cases and the advantages of larger dictionaries can be nearly fully recovered.

Lastly, the convergence speed of the LCA operation is analyzed for different R_{line} values. Figure 6-11(a) plots the MSE of the reconstructed image (lower panels) during the dynamic evolution of the memristor network when performing LCA analysis. Two regions, an abrupt MSE reduction at the early part, followed by much gradual and slow MSE changes, can be observed. In particular, the slope of the abrupt change region becomes steeper as R_{line} increases, as shown in the inset of Figure 6-11(a). To gain insight of this behavior, the neuron dynamics (upper panel) and MSE during LCA analysis of a single patch is examined, as shown in Figure 6-11(b). First, the region of the abrupt MSE reduction exactly matches with the region where the neurons' activities show an oscillating behavior, while the slower MSE reduction correspond to the gradual evolution of the neuron activities. Moreover, as R_{line} increases, the neuron oscillations occur in fewer iterations and accordingly the MSE drops more quickly. The neuron oscillation is caused by the initial over-representation of the original image due to simultaneous activation of multiple similar neurons in a large size dictionary, followed by quick drop of the membrane potential in the next cycle of LCA due to the inhibition effect shown in Eq. (6-7). Overtime a sparse representation of the input is obtained and the oscillation behaviors stop. In this sense, a high R_{line} degrades the neuron outputs non-uniformly and thus suppresses multiple neurons firing together, leading to

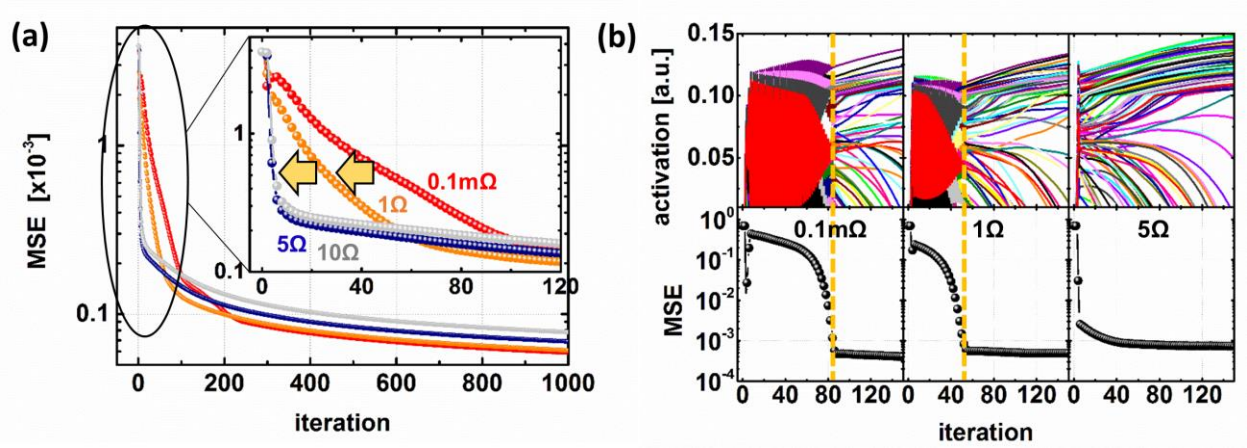

Figure 6-11: The evolution of the MSE during iteration. (a) The MSE decreases in two steps: abrupt at the early stage and following gradual changes. In particular, larger R_{line} makes even rapid changes in the abrupt area as zoom-in plot (inset). (b) The evolution of the neuron dynamics and the MSE from analyzing a single patch. The region of the abrupt MSE reduction corresponds to the region where large oscillations of the neuron membrane potentials are observed. Higher R_{line} suppresses the number of activated neurons and reduces the oscillations, leading to faster convergence but an overall higher MSE.

faster convergence in the MSE. Although we note the absolute MSE obtained with high R_{line} is still higher than that obtained in the low R_{line} case. In the same sense, smaller array size and higher threshold λ will also reduce the oscillation and iteration numbers during LCA analysis, but result in higher overall MSE after network stabilization.

6.5 Conclusion

In this chapter we discussed parasitic effects, such as series resistance and sneak path, on the operation of memristor crossbar based neuromorphic operations. Although the sneak path leakage effect is negligible during dot product operations, line resistance effects can significantly distort the output current non-uniformly and affect the activities of the output neurons and how the winning neurons are identified. Additionally, R_{wl} shows worse impact on the forward dot product

operation than R_{bl}. A simplified numerical model that includes only simple, known parameters was proposed and was shown to effectively compensate the parasitic effects both during the single dot product operation and during complete neuromorphic operations including training and inference. A sparse-coding based feature extraction algorithm was analyzed in detail, and the series resistance effect, if not properly treated, could prevent the network from being sufficiently trained with training concentrated only in small numbered neurons, and significantly degrade the LCA accuracy. The simple scaling method was found to be able to effectively compensate the parasitic effects both during learning and during inference, and restore the desired outputs. Considering parasitic effects are difficult to be completely avoided in hardware implementation of neuromorphic systems, the analysis presented in this study provides practical solutions to help mitigate the parasitic effects and will help bring such hardware systems to reality.

Chapter 7. Future Work

We have discussed several forms of first-order and second-order memristors and their applications, including issues and possible solutions for practical network implementations. In this chapter, we will discuss several future projects that aim to continue optimize the device structure and fully utilize the devices for neuromorphic applications.

7.1 Device Optimization

For neuromorphic applications, a memristor should have characteristics such that its size, energy consumption, operation speed, and dynamic range are comparable or even better than its biological counterparts. In this regard, the tantalum oxide based memristors we have developed needs further optimization.

For example, a larger range that offers linear analog conductance change is beneficial during online learning. In chapter 4.2, we have shown an approach using multiple state variables to improve the analog conductance range. However, in general studies focusing on the analog property of the memristor are still limited.

Most of reported studies have targeted more gradual SET operations, since the SET process typically results in digital-type (abrupt) conductance changes originating from a positive feedback during the process, due to increased electric field and elevated temperature as the filament grows , while a reset process can be gradual [117]–[120]. Given the previous reports arguing that weak and multiple filaments instead of a single strong conductive path is desirable to achieve the analog behavior [121], slowing down the ion migration speed and creating more uniform oxygen vacancy

(V_Os) distribution may provide better analog property. Al (or AlOx) is considered as one of the materials able to achieve such an effect [122]–[124]. V_Os movement in AlOx is much slower than in other typical transition metal oxides (TMOs) used in memristors such as HfOx and TaOx [118]. Moreover, the higher bandgap of AlOx can create a tunnel barrier between the switching layer and the electrode and therefore prevents strong filament formation by reducing current overshoot during the forming process [125]. Additionally, Al doping in TMO can reduce the formation energy of V_Os and enable more uniformly distributed V_Os near the Al dopants [126]. Thus, Al doping (or a thin AlOx layer) may be employed to further improve the analog characteristic of the TaOx-based memristor we used.

Specifically, the bottom electrode can be replaced from Pd/Au to Al/Pd/Au or Al/Au to form a thin AlOx layer at the interface with the switching layer such as Ta_2O_{5-x}. Another approach is to use an ALD tool to obtain AlOx films with controlled density and quality by controlling the deposition temperature, water dosing and plasma, such that a high-density AlOx layer can be used to prevent overshoot, while a low-density film can serve as the switching layer with slow V_Os migration speed as mentioned.

7.2 Convolution Neural Network (CNN) Mapping with Memristor Array

The development of 32×12 sized arrays as described in chapter 5 will make it possible to implement more complex tasks such as convolution neural networks (CNNs), where massive VMM operations are employed during both training and testing. From a simple simulation using the TensorFlow API, relatively high, over than 95% of classification accuracy can be obtained even in a small two-layer network, with hidden layers of 25×8 and 32×10, for MNIST dataset classification, as shown in Figure 7-1. Moreover, in simulations based on offline learning, the CNN

1st array input = filter size	1st array output = number of filters	2nd array input		accuracy
		pooling size	number of max filters	
max 5x5	max 12	7x7	max 2	93.40%
		14x14	max 8	95.16%

Figure 7-1: Simulation results from TensorFlow using the MNSIT dataset. A small two-layer network can already achieve over 95% of classification accuracy.

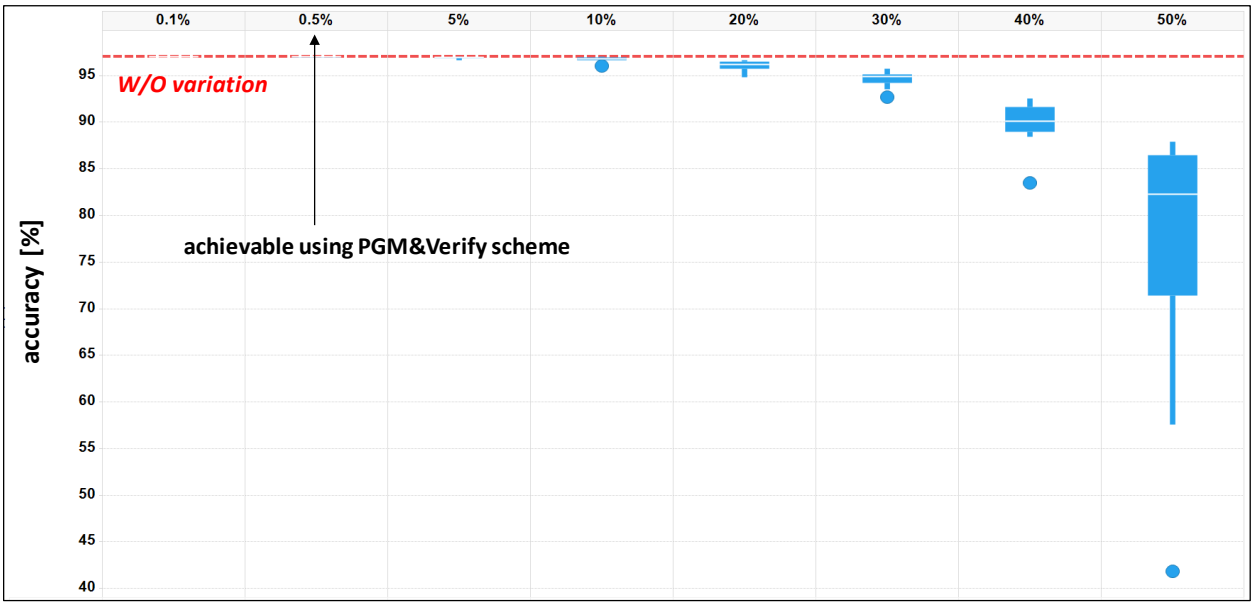

Figure 7-2: Dependence of offline-based CNN classification accuracy on device variation. The classification accuracy is not affected up to 5% of variation, whereas optimized Ta_2O_5 devices exhibit only 0.5% variation.

network can tolerate up to 5% of device variation without degradation of accuracy, as shown in Figure 7-2, while our optimized Ta_2O_5 devices exhibit only ~0.5% variation as mentioned in chapter 2. Thus, mapping of multilayer CNN networks on memristor arrays can be a promising future work.

7.3 Improving Second-order Characteristics and Network Functionalities

Second-order memristors can natively mimic biological neural systems, as discussed in Chapter 4. The ability to control the short-term time constant is essential to allow the practical operations of the second-order memristors. In our case, the internal temperature dynamics was used as a second state variable, so heat dissipation is a critical process to be looked into. The top electrode material can affect heat dissipation within the switching layer and could be further explored. New device structures, for example, a heat insulating layer or a buffer layer could be added to control the heat dissipation.

On the other hand, emulation of newly discovered concepts in biology such as the neuromordulation effect could be a promising future work to enable network functionalities in second-order memristor systems. With recent elaborate experimental studies, chemicals that act as neurotransmitters, such as dopamine, serotonin, acetylcholine, noradrenaline and histamine, are sub-categorized as neuromodulators according to their functions, release manner, and target receptors [127]–[131]. The neuromodulator, a subset of neurotransmitter, is released by means of volume transmission, which results in a wide range of influence on the global neural tissues, compared with conventional localized neurotransmitters, as shown in Figure 7-3. Recent studies further discovered the role of neuromodulators on STDP. First, neuromodulators serve as a global gate for triggering STDP in several brain regions such as the dorsal striatum [127], [132], [133]. In other words, without the presence of neuromodulator synaptic efficacy would not change even if the event of pre- and post-spikes takes place closely, e.g. within a few *m*sec. Second, neuromodulators not only enables STDP, they are critical to tune the window and shape of STDP [127], [134], [135]. For example, dopamine and noradrenaline in hippocampal neurons widen the LTP window and change the time range required for LTP induction. These experimental evidences

proved the theoretical concept that a "third-factor operation" [136] is essential to decide and regulate STDP, in contrast to the conventional theory that relies only on the activity of classical neurotransmitters induced by the arrival of pre- and pose-spikes.

Incorporating such a novel bio-realistic synaptic behavior into solid-state devices can expand the functionalities of the memristor network. Further research at both the single device level and network level is needed to demonstrate such a faithfully bio-inspired neuromorphic computing system.

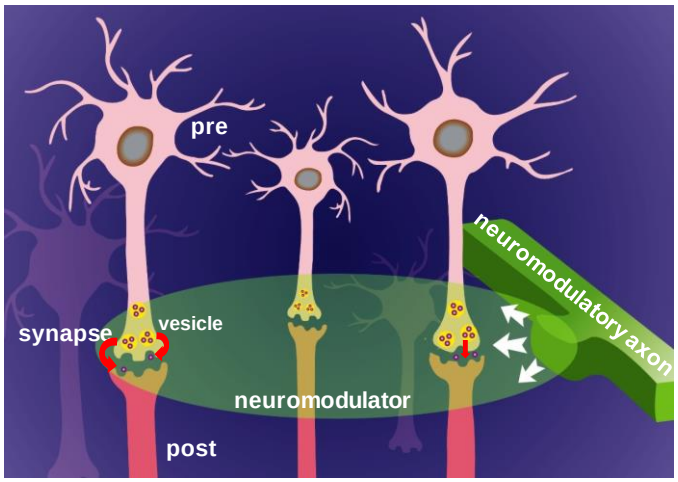

Figure 7-3: Schematic of the nervous system including synapses and neurons, with an emphasis on the working range of the neuromodulator, where multiple synapses within a wide area undergo the synaptic changes, in contrast to conventional theories .

Appendix

Supplementary Material

8.1 Device Model

The switching characteristics of Ta_2O_5 memristors can be fitted with a memristor device model first developed in ref. [59], [137].

$$I = w\gamma \sinh(\delta \times V) + (1-w)\alpha\left(1 - e^{-\beta \times V}\right) \qquad \text{(S1)}$$

$$= w\left\{\gamma \sinh(\delta \times V) - \alpha\left(1 - e^{-\beta \times V}\right)\right\} + \alpha\left(1 - e^{-\beta \times V}\right)$$

$$\frac{dw}{dt} = (1-w)^2\kappa(e^{\mu_1 V} - e^{-\mu_2 V})u(V) + w^2\kappa(e^{\mu_1 V} - e^{-\mu_2 V})u(-V) \qquad \text{(S2)}$$

, where equation (S1) is the current-voltage equation that depends on the internal state variable *w*, and equation (S2) describes the dynamics of the state variable. In our device the internal state variable *w* represents the effective area of the conductive channel. γ, δ, α, and β in equation (S1) represent the effective tunneling distance and the tunneling barrier in the channel region, the depletion width and the Schottky barrier height in the Schottky barrier region, respectively. κ, μ_1, and μ_2, in equation (S2) are fitting parameters related to the ion hopping process that modulates the channel width *w*. *u*() is the Heaviside step function that accounts for the bias polarity dependency in write and erase operations.

8.2 Fabrication Recipe

The Ta_2O_5-based memristor networks used in our studies were fabricated in a crossbar structure in which memristive devices are formed at each cross-points. On a SiO_2 (100 nm) / Si substrate, the bottom Pd electrode (50 nm) with an adhesion layer of NiCr (5 nm) was deposited by e-beam evaporation and patterned with a width of 2 um by photolithography (GCA AS200 Autostep) and lift-off processes. Then, the Ta_2O_{5-x} film (3.5 nm) was deposited by radio frequency (RF) sputtering with very small power (30 W) using a Ta_2O_5 ceramic target at room temperature. Next, the Ta and Pd/Au top electrode (40 and 40/100 nm) was deposited by direct current (DC) sputtering with power of 100 W and e-beam evaporation, respectively, followed by patterning processes in same manner with the bottom electrode. Finally, the bottom electrode pads were opened by etching the Ta_2O_{5-x} layer using a reactive ion etching (RIE) process with SF6/Ar gases.

8.3 Board System

In the *K*-means and PDE experiments, a custom-built test board was used to program the memristors in the array and to perform the VMM operations. The memristor array was wire-bonded to a chip carrier and plugged in the board. The board can apply voltage pulses to the proper rows and columns and measure output currents of given addresses. Two digital-to-analog converters (DACs) as voltage sources (from -5V to +5V) are connected to rows and columns of a memristor array respectively (total 4 DACs) through switching matrix for addressing (Supplementary5). DAC0 (DAC2) is connected to selected rows (columns) and DAC1 (DAC3) is for remaining unselected rows (columns). For a read operation, output node is virtually grounded with negative feedback by Op. Amp. Instead of connected to DAC4 and a 1KΩ resistor converts the current to a voltage that can be read by analog-to-digital converters (ADCs). The board is then connected to a microcontroller implemented on the FPGA (Xilinx, Spartan6) on the test board and

the algorithm is executed in a fully automated manner by Python programming. Through above configuration, the board can perform array-based operations including random read and write (erase) by DC sweep or pulse measurement.

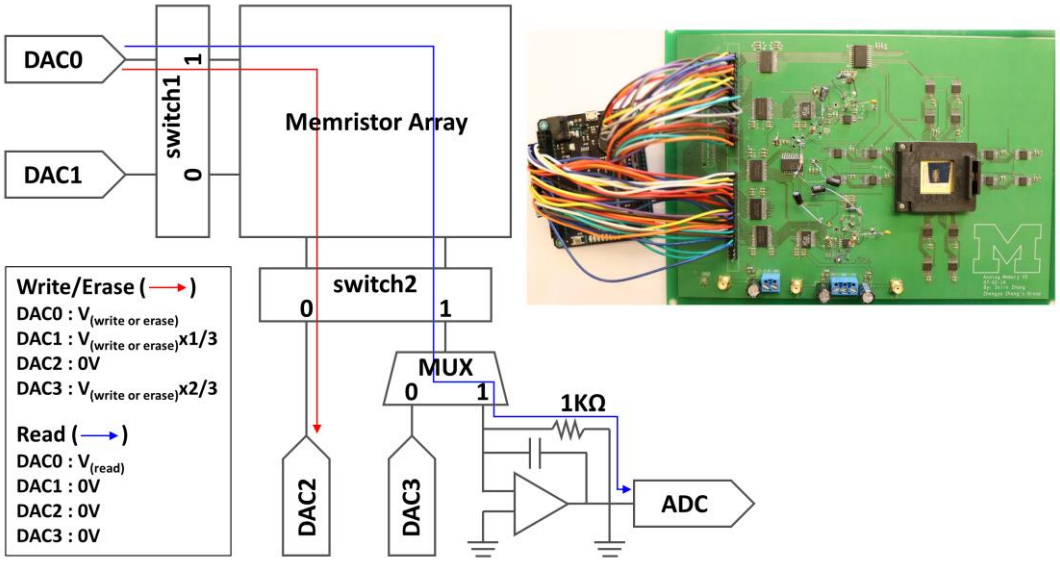

Figure S 1: Block diagram of the test board and photo of the test board with an integrated memristor chip.

Bibliography

[1] C. Mead, “Neuromorphic Electronic Systems,” *Proc. IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[2] R. Sarpeshkar, “Analog Versus Digital: Extrapolating from Electronics to Neurobiology,” *Neural Comput.*, vol. 10, no. 7, pp. 1601–1638, 1998.

[3] G. Indiveri and S. C. Liu, “Memory and Information Processing in Neuromorphic Systems,” *Proc. IEEE*, vol. 103, no. 8, pp. 1379–1397, 2015.

[4] G. E. Moore, “Cramming more components onto integrated circuits,” *Electronics*, vol. 38, pp. 114–117, 1965.

[5] R. R. Schaller, “MOORE ’ S LAW : past , present ,” *IEEE Spectr.*, vol. 34, pp. 52–59, 1997.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances In Neural Information Processing Systems*, 2012, pp. 1106–1114.

[7] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] D. Silver *et al.*, “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[9] G. Brumfiel, “High-energy physics: Down the petabyte highway,” *Nature*, vol. 469, no. 7330, pp. 282–283, 2011.

[10] E. Santos *et al.*, “UV-CDAT: Analyzing climate datasets from a user’s perspective,”

Comput. Sci. Eng., vol. 15, no. 1, pp. 94–103, 2013.

[11] J. Fan, F. Han, and H. Liu, "Challenges of Big Data analysis," *Natl. Sci. Rev.*, vol. 1, no. 2, pp. 293–314, 2014.

[12] Y. Achdou, F. J. Buera, J.-M. Lasry, P.-L. Lions, and B. Moll, "Partial differential equation models in macroeconomics," *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 372, no. 2028, p. 20130397, 2014.

[13] P. Bauer, A. Thorpe, and G. Brunet, "The quiet revolution of numerical weather prediction," *Nature*, vol. 525, no. 7567, pp. 47–55, 2015.

[14] J. Backus, "Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs," *Commun. ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[15] H. Markram, "The human brain project," *Sci. Am.*, vol. 306, pp. 50–55, 2012.

[16] P. A. Merolla *et al.*, "Amillion spiking-neuron integrated circuitwith a scalable communication network and interface," *Science (80-.).*, vol. 345, no. 7812, pp. 668–673, 2014.

[17] N. Qiao *et al.*, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Front. Neurosci.*, vol. 9, no. APR, pp. 1–17, 2015.

[18] K. Boahen, "A Neuromorph's Prospectus," *Comput. Sci. Eng.*, vol. 19, no. 2, pp. 14–28, 2017.

[19] L. O. Chua, "Memristor—The Missing Circuit Element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[20] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[21] R. Waser and M. Aono, “Nonoionics-based resistive switching memories,” *Nat. Mater.*, vol. 6, p. 833, 2007.

[22] T. W. Hickmott, “Low-frequency negative resistance in thin anodic oxide films,” *J. Appl. Phys.*, vol. 33, no. 9, pp. 2669–2682, 1962.

[23] I. G. Baek *et al.*, “Highly scalable non-volatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses,” in *IEDM Technical Digest. IEEE International Electron Devices Meeting,* 2004, pp. 587–590.

[24] S. Seo *et al.*, “Reproducible resistance switching in polycrystalline NiO films,” *Appl. Phys. Lett.*, vol. 85, no. 23, pp. 5655–5657, 2004.

[25] C. Rohde, B. J. Choi, D. S. Jeong, S. Choi, J. S. Zhao, and C. S. Hwang, “Identification of a determining parameter for resistive switching of TiO2 thin films,” *Appl. Phys. Lett.*, vol. 86, no. 26, pp. 1–3, 2005.

[26] J. J. Yang *et al.*, “High switching endurance in TaOxmemristive devices,” *Appl. Phys. Lett.*, vol. 97, p. 232102, 2010.

[27] A. C. Torrezan, J. P. Strachan, G. Medeiros-Ribeiro, and R. S. Williams, “Sub-nanosecond switching of a tantalum oxide memristor,” *Nanotechnology*, vol. 22, p. 485203, 2011.

[28] M. J. Lee *et al.*, “A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2O5-xx/TaO2-xbilayer structures,” *Nat. Mater.*, vol. 10, no. 8, pp. 625–630, 2011.

[29] S. Choi, J. Lee, S. Kim, and W. D. Lu, “Retention failure analysis of metal-oxide based resistive memory,” *Appl. Phys. Lett.*, vol. 105, p. 113510, 2014.

[30] B. J. Choi *et al.*, “High-Speed and Low-Energy Nitride Memristors,” *Adv. Funct. Mater.*,

vol. 26, no. 29, pp. 5290–5296, 2016.

[31] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, “Nanoscale memristor device as synapse in neuromorphic systems,” *Nano Lett.*, vol. 10, no. 4, pp. 1297–1301, 2010.

[32] T. Ohno, T. Hasegawa, T. Tsuruoka, K. Terabe, J. K. Gimzewski, and M. Aono, “Short-term plasticity and long-term potentiation mimicked in single inorganic synapses,” *Nat. Mater.*, vol. 10, no. 8, pp. 591–595, 2011.

[33] D. Kuzum, R. G. D. Jeyasingh, B. Lee, and H. S. P. Wong, “Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing,” *Nano Lett.*, vol. 12, no. 5, pp. 2179–2186, 2012.

[34] Y. V. Pershin and M. Di Ventra, “Experimental demonstration of associative memory with memristive neural networks,” *Neural Networks*, vol. 23, no. 7, pp. 881–886, 2010.

[35] J. J. Yang, D. B. Strukov, and D. R. Stewart, “Memristive devices for computing,” *Nat. Nanotechnol.*, vol. 8, no. 1, pp. 13–24, 2013.

[36] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, “Training and operation of an integrated neuromorphic network based on metal-oxide memristors,” *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[37] P. M. Sheridan, F. Cai, C. Du, W. Ma, Z. Zhang, and W. D. Lu, “Sparse coding with memristor networks,” *Nat. Nanotechnol.*, vol. 12, no. 8, pp. 784–789, 2017.

[38] M. Hu *et al.*, “Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication,” in *IEEE Design Automation Conference*, 2016, p. 19.

[39] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, *Principles of neural science*, 4th ed. New

York: McGraw-hill, 2000.

[40] S. Larentis, F. Nardi, S. Balatti, D. C. Gilmer, and D. Ielmini, “Resistive Switching by Voltage-Driven Ion Migration in Bipolar RRAM-Part II: Modeling,” *IEEE Trans. Electron Devices*, vol. 59, no. 9, pp. 2468–2475, 2012.

[41] S. Kim, S. Choi, and W. Lu, “Comprehensive physical model of dynamic resistive switching in an oxide memristor,” *ACS Nano*, vol. 8, no. 3, pp. 2369–2376, 2014.

[42] S. Kim, C. Du, P. Sheridan, W. Ma, S. Choi, and W. D. Lu, “Experimental demonstration of a second-order memristor and its ability to biorealistically implement synaptic plasticity,” *Nano Lett.*, vol. 15, no. 3, pp. 2203–2211, 2015.

[43] B. Y. V Pershin and M. Di Ventra, “Quantum Computation With Memory Circuit Elements,” *Proc. IEEE*, vol. 100, no. 6, pp. 1–10, 2011.

[44] G. Q. Bi and M. M. Poo, “Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type.,” *J. Neurosci.*, vol. 18, no. 24, pp. 10464–10472, 1998.

[45] S. Ben Achour and O. Pascual, “Glia: The many ways to modulate synaptic plasticity,” *Neurochem. Int.*, vol. 57, no. 4, pp. 440–445, 2010.

[46] H. Markram, W. Gerstner, and P. J. Sjöström, “A history of spike-timing-dependent plasticity,” *Front. Synaptic Neurosci.*, vol. 3, no. AUG, pp. 1–24, 2011.

[47] S. Yang, Y. Tang, and R. Zucker, “Selective induction of LTP and LTD by postsynaptic i elevation,” *J. Neurophysiol.*, vol. 81, pp. 781–787, 1999.

[48] H. Z. Shouval, M. F. Bear, and L. N. Cooper, “A unified model of NMDA receptor-dependent bidirectional synaptic plasticity,” *Proc. Natl. Acad. Sci.*, vol. 99, no. 16, pp. 10831–10836, 2002.

[49] J. W. Rudy, *The neurobiology of learning and memory*. Sunderland, MA: Sinauer Associates, Inc., 2008.

[50] Y. Yao *et al.*, "PKM Maintains Late Long-Term Potentiation by N-Ethylmaleimide-Sensitive Factor/GluR2-Dependent Trafficking of Postsynaptic AMPA Receptors," *J. Neurosci.*, vol. 28, no. 31, pp. 7820–7827, 2008.

[51] C. Du, W. Ma, T. Chang, P. Sheridan, and W. D. Lu, "Biorealistic Implementation of Synaptic Functions with Oxide Memristors through Internal Ionic Dynamics," *Adv. Funct. Mater.*, vol. 25, no. 27, pp. 4290–4299, 2015.

[52] S. G. Hu *et al.*, "Associative memory realized by a reconfigurable memristive Hopfield neural network," *Nat. Commun.*, vol. 6, no. May, pp. 1–5, 2015.

[53] G. E. Hinton and T. J. Sejnowski, *Unsupervised learning: foundations of neural computation*. MIT press, 1999.

[54] I. Goodfellow *et al.*, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 2672–2680.

[55] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science (80-.).*, vol. 313, no. July, pp. 504–508, 2006.

[56] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, 1967, pp. 281–297.

[57] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1–3, pp. 1–6, 1998.

[58] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta, "A survey of kernel and spectral methods for clustering," *Pattern Recognit.*, vol. 41, no. 1, pp. 176–190, 2008.

[59] S. Choi, J. H. Shin, J. Lee, P. Sheridan, and W. D. Lu, "Experimental Demonstration of

Feature Extraction and Dimensionality Reduction Using Memristor Networks," *Nano Lett.*, vol. 17, no. 5, pp. 3113–3118, 2017.

[60] Y. Jiang, J. Kang, and X. Wang, "RRAM-based parallel computing architecture using k-nearest neighbor classification for pattern recognition," *Sci. Rep.*, vol. 7, no. December 2016, pp. 1–8, 2017.

[61] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.

[62] Y. Bengio, *Learning deep architectures for AI*. Now Publishers, Inc., 2009.

[63] D. Lin and X. Wu, "Phrase clustering for discriminative learning," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - ACL-IJCNLP '09*, 2009, vol. 2, no. August, p. 1030.

[64] A. Coates, A; Ng, *Neural Networks: Tricks of the Trade*. Berlin: Springer, 2012.

[65] S. Yu, B. Gao, Z. Fang, H. Yu, J. Kang, and H. S. P. Wong, "A low energy oxide-based electronic synaptic device for neuromorphic visual systems with tolerance to device variation," *Adv. Mater.*, vol. 25, no. 12, pp. 1774–1779, 2013.

[66] P. M. Sheridan, C. Du, and W. D. Lu, "Feature Extraction Using Memristor Networks," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 27, no. 11, pp. 2327–2336, 2016.

[67] G. H. Baek, A. R. Lee, T. Y. Kim, H. S. Im, and J. P. Hong, "Oxide stoichiometry-controlled TaOx-based resistive switching behaviors," *Appl. Phys. Lett.*, vol. 109, no. 14, p. 143502, 2016.

[68] B. Gao *et al.*, "A novel defect-engineering-based implementation for high-performance multilevel data storage in resistive switching memory," *IEEE Trans. Electron Devices*,

vol. 60, no. 4, pp. 1379–1383, 2013.

[69] K. Bache and M. Lichman, “UCI machine learning repository,” *University of California Irvine*, 2013. [Online]. Available: http://archive.ics.uci.edu/ml.

[70] “Modeling and simulation at the exascale for energy and the environment,” 2007.

[71] T. Palmer, “Build_imprecise supercomputers,” *Nature*, vol. 526, pp. 32–33, 2015.

[72] N. Aage, E. Andreassen, B. S. Lazarov, and O. Sigmund, “Giga-voxel computational morphogenesis for structural design,” *Nature*, vol. 550, no. 7674, pp. 84–86, 2017.

[73] P. M. Altrock, L. L. Liu, and F. Michor, “The mathematics of cancer: Integrating quantitative models,” *Nat. Rev. Cancer*, vol. 15, no. 12, pp. 730–745, 2015.

[74] M. A. Zidan, A. Chen, G. Indiveri, and W. D. Lu, “Memristive computing devices and applications,” *J. Electroceramics*, vol. 39, no. 1–4, pp. 4–20, 2017.

[75] S. Yu, P. Y. Chen, Y. Cao, L. Xia, Y. Wang, and H. Wu, “Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect,” *Tech. Dig. - Int. Electron Devices Meet. IEDM*, vol. 2016–Febru, p. 17.3.1-17.3.4, 2015.

[76] E. O. Neftci, B. U. Pedroni, S. Joshi, M. Al-Shedivat, and G. Cauwenberghs, “Stochastic synapses enable efficient brain-inspired learning machines,” *Front. Neurosci.*, vol. 10, no. JUN, pp. 1–16, 2016.

[77] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, “High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm,” *Nanotechnology*, vol. 23, no. 7, pp. 3–10, 2012.

[78] C. Li *et al.*, “Analogue signal and image processing with large memristor crossbars,” *Nat. Electron.*, vol. 1, no. January, p. 52, 2018.

[79] D. Gilbarg and N. Trudinger, *Elliptic partial differential equations of second order*.

springer, 2015.

[80] W. F. Ames, *Numerical methods for partial differential equations*. Academic press, 2014.

[81] Y. Nishidate and G. P. Nikishkov, "Fast Water Animation Using the Wave Equation with Damping," in *Internationa lConference on Computational Science*, 2005, pp. 232–239.

[82] M. Zidan, Y. J. Jeong, J. H. Shin, C. Du, Z. Zhang, and W. Lu, "Field-Programmable Crossbar Array (FPCA) for Reconfigurable Computing," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 7766, no. c, pp. 1–13, 2017.

[83] M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," *Nat. Electron.*, vol. 1, no. 1, pp. 22–29, 2018.

[84] Y. J. Jeong, S. Kim, and W. D. Lu, "Utilizing multiple state variables to improve the dynamic range of analog switching in a memristor," *Appl. Phys. Lett.*, vol. 107, no. 17, pp. 10–15, 2015.

[85] N. F. Mott and R. W. Gurney, *Electronic processes in ionic crystals*. Oxford University Press, 1948.

[86] M. A. Zidan, Y. J. Jeong, and W. D. Lu, "Temporal Learning Using Second-Order Memristors," *IEEE Trans. Nanotechnol.*, vol. 16, no. 4, pp. 721–723, 2017.

[87] W. Ma, F. Cai, C. Du, Y. Jeong, M. Zidan, and W. D. Lu, "Device nonideality effects on image reconstruction using memristor arrays," in *Technical Digest - International Electron Devices Meeting, IEDM*, 2017, p. 16.7.1-16.7.4.

[88] A. Flocke and T. G. Noll, "Fundamental analysis of resistive nano-crossbars for the use in hybrid nano/CMOS-memory," in *ESSCIRC 2007 - Proceedings of the 33rd European Solid-State Circuits Conference*, 2007, pp. 328–331.

[89] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based

memory: The sneak paths problem and solutions," *Microelectronics J.*, vol. 44, no. 2, pp. 176–183, 2013.

[90] J. Zhou, S. Member, K. Kim, and W. Lu, "Crossbar RRAM Arrays : Selector Device Requirements During Read Operation," vol. 61, no. 5, pp. 1369–1376, 2014.

[91] S. Kim, S. Choi, J. Lee, and W. D. Lu, "Tuning resistive switching characteristics of tantalum oxide memristors through Si doping," *ACS Nano*, vol. 8, no. 10, pp. 10262–10269, 2014.

[92] K. Seo *et al.*, "Analog memory and spike-timing-dependent plasticity characteristics of a nanoscale titanium oxide bilayer resistive switching device," *Nanotechnology*, vol. 22, no. 25, p. 254023, 2011.

[93] V. Kornijcuk *et al.*, "Multiprotocol-induced plasticity in artificial synapses," *Nanoscale*, vol. 6, no. 24, pp. 15151–15160, 2014.

[94] H. Mähne, H. Wylezich, F. Hanzig, S. Slesazeck, D. Rafaja, and T. Mikolajick, "Analog resistive switching behavior of Al/Nb2O5/Al device," *Semicond. Sci. Technol.*, vol. 29, no. 10, p. 104002, 2014.

[95] H. J. Kim *et al.*, "Digital versus analog resistive switching depending on the thickness of nickel oxide nanoparticle assembly," *RSC Adv.*, vol. 3, no. 43, p. 20978, 2013.

[96] K. Moon *et al.*, "Hardware implementation of associative memory characteristics with analogue-type resistive-switching device," *Nanotechnology*, vol. 25, no. 49, p. 495204, 2014.

[97] J. D. Kim *et al.*, "Investigation of analog memristive switching of iron oxide nanoparticle assembly between Pt electrodes," *J. Appl. Phys.*, vol. 114, no. 22, p. 224505, 2013.

[98] C. Wang, W. He, Y. Tong, and R. Zhao, "Investigation and Manipulation of Different

Analog Behaviors of Memristor as Electronic Synapse for Neuromorphic Applications," *Sci. Rep.*, vol. 6, no. February, pp. 1–9, 2016.

[99] S. Kim, J. Zhou, and W. D. Lu, "Crossbar RRAM arrays: Selector device requirements during write operation," *IEEE Trans. Electron Devices*, vol. 61, no. 8, pp. 2820–2826, 2014.

[100] E. H. Sondheimer, *The mean free path of electrons in metals*, vol. 1, no. 1. 1952.

[101] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nat. Mater.*, vol. 9, no. 5, pp. 403–406, 2010.

[102] G. Schindler, G. Steinlesberger, M. Engelhardt, and W. Steinhögl, "Electrical characterization of copper interconnects with end-of-roadmap feature sizes," *Solid. State. Electron.*, vol. 47, no. 7 SPEC., pp. 1233–1236, 2003.

[103] W. Steinhögl *et al.*, "Tungsten interconnects in the nano-scale regime," *Microelectron. Eng.*, vol. 82, no. 3–4 SPEC. ISS., pp. 266–272, 2005.

[104] M. E. Fouda, A. M. Eltawil, and F. Kurdahi, "Modeling and Analysis of Passive Switching Crossbar Arrays," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 65, no. 1, pp. 270–282, 2018.

[105] M. Oljaca and B. Baker, "How the voltage reference affects ADC performance, Part 1," *Analog Appl. J.*, vol. 2Q, pp. 5–8, 2009.

[106] B. Y. D. H. Hubel, a D. T. N. Wiesel, D. N. Hubel, T. N. Wiesel, B. Y. D. H. Hubel, and a D. T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.*, vol. 160, no. 1, pp. 106–154, 1962.

[107] T. Serre, L. Wolf, and T. Poggio, "Object recognition with features inspired by visual cortex," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2, pp. 994–

1000, 2005.

[108] O. Bichler, D. Querlioz, S. J. Thorpe, J. P. Bourgoin, and C. Gamrat, “Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity,” *Neural Networks*, vol. 32, pp. 339–348, 2012.

[109] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, “STDP and sTDP variations with memristors for spiking neuromorphic learning systems,” *Front. Neurosci.*, vol. 7, no. 7 FEB, pp. 1–15, 2013.

[110] E. Oja, “Simplified neuron model as a principal component analyzer,” *J. Math. Biol.*, vol. 15, no. 3, pp. 267–273, 1982.

[111] T. D. Sanger, “Optimal unsupervised learning in a single-layer linear feedforward neural network,” *Neural Networks*, vol. 2, no. 6, pp. 459–473, 1989.

[112] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, “Sparse Coding via Thresholding and Local Competition in Neural Circuits,” *Neural Comput.*, vol. 20, no. 10, pp. 2526–2563, 2008.

[113] W. Woods, J. Bürger, and C. Teuscher, “Synaptic Weight States in a Locally Competitive Algorithm for Neuromorphic Memristive Hardware,” *IEEE Trans. Nanotechnol.*, vol. 14, no. 6, pp. 945–953, 2015.

[114] B. A. Olshausen and D. J. Field, “Sparse coding with an overcomplete basis set: A strategy employed by V1?,” *Vision Res.*, vol. 37, no. 23, pp. 3311–3325, 1997.

[115] R. Mehrotra, K. R. Namuduri, and N. Ranganathan, “Gabor filter-based edge detection,” *Pattern Recognit.*, vol. 25, no. 12, pp. 1479–1494, 1992.

[116] T. E. Weldon, W. E. Higgins, D. E. Dunn, and D. E. D. Thomas E. Weldon William E. Higgins, “Efficient Gabor Filter Design for Texture Segmentation,” *Pattern Recognit.*,

vol. 29, no. 12, pp. 2005–2015, 1996.

[117] B. Gao, H. Wu, J. Kang, H. Yu, and H. Qian, “Oxide-based analog synapse: Physical modeling, experimental characterization, and optimization,” in *Electron Devices Meeting (IEDM), 2016 IEEE International*, 2016, p. 7.3. 1-7.3. 4.

[118] J. Woo *et al.*, “Improved synaptic behavior under identical pulses using AlOx/HfO2bilayer RRAM array for neuromorphic systems,” *IEEE Electron Device Lett.*, vol. 37, no. 8, pp. 994–997, 2016.

[119] L. Larcher, A. Padonavi, and Va. Di Lecce, “Multiscale modeling of neuromorphic computing: from materials to device operations,” 2017, pp. 282–285.

[120] J. Woo, K. Moon, J. Song, M. Kwak, J. Park, and H. Hwang, “Optimized Programming Scheme Enabling Linear Potentiation in Filamentary HfO2RRAM Synapse for Neuromorphic Systems,” *IEEE Trans. Electron Devices*, vol. 63, no. 12, pp. 5064–5067, 2016.

[121] J. Woo, A. Padovani, K. Moon, M. Kwak, L. Larcher, and H. Hwang, “Linking Conductive Filament Properties and Evolution to Synaptic Behavior of RRAM Devices for Neuromorphic Applications,” *IEEE Electron Device Lett.*, vol. 38, no. 9, pp. 1220–1223, 2017.

[122] K. Seo *et al.*, “Analog memory and spike-timing-dependent plasticity characteristics of a nanoscale titanium oxide bilayer resistive switching device,” *Nanotechnology*, vol. 22, no. 25, 2011.

[123] X. Li *et al.*, “Electrode-induced digital-to-analog resistive switching in TaO x -based RRAM devices,” *Nanotechnology*, vol. 27, no. 30, p. 305201, 2016.

[124] M. Alayan *et al.*, “Self-rectifying behavior and analog switching under identical pulses

using Tri-layer RRAM crossbar array for neuromorphic systems," in *2017 IEEE 9th International Memory Workshop, IMW 2017*, 2017, pp. 1–4.

[125] S. Kim, H. Kim, S. Hwang, M. H. Kim, Y. F. Chang, and B. G. Park, "Analog Synaptic Behavior of a Silicon Nitride Memristor," *ACS Appl. Mater. Interfaces*, vol. 9, no. 46, pp. 40420–40427, 2017.

[126] B. Gao *et al.*, "Modeling Disorder Effect of the Oxygen Vacancy Distribution in Filamentary Analog RRAM for Neuromorphic Computing," in *IEDM, Technical Digest - International Electron Devices Meeting*, 2017, pp. 91–94.

[127] V. Pawlak, J. R. Wickens, A. Kirkwood, and J. N. D. Kerr, "Timing is not everything: Neuromodulation opens the STDP gate," *Front. Synaptic Neurosci.*, vol. 2, no. OCT, pp. 1–14, 2010.

[128] W. Schultz, "Getting formal with dopamine and reward," *Neuron*, vol. 36, no. 2, pp. 241–263, 2002.

[129] A. J. Yu and P. Dayan, "Uncertainty, neuromodulation, and attention," *Neuron*, vol. 46, no. 4, pp. 681–692, 2005.

[130] M. R. Picciotto, M. J. Higley, and Y. S. Mineur, "Acetylcholine as a Neuromodulator: Cholinergic Signaling Shapes Nervous System Function and Behavior," *Neuron*, vol. 76, no. 1, pp. 116–129, 2012.

[131] J. O'Donnell, D. Zeppenfeld, E. McConnell, S. Pena, and M. Nedergaard, "Norepinephrine: A neuromodulator that boosts the function of multiple cell types to optimize CNS performance," *Neurochem. Res.*, vol. 37, no. 11, pp. 2496–2512, 2012.

[132] S. Bissieére, Y. Humeau, and A. Luüthi, "Dopamine gates LTP induction in lateral amygdala by suppressing feedforward inhibition," *Nat. Neurosci.*, vol. 6, no. 6, pp. 587–

592, 2003.

[133] V. Pawlak and J. N. D. Kerr, “Dopamine Receptor Activation Is Required for Corticostriatal Spike-Timing-Dependent Plasticity,” *J. Neurosci.*, vol. 28, no. 10, pp. 2435–2446, 2008.

[134] Y.-W. Lin, M.-Y. Min, T.-H. Chiu, and H.-W. Yang, “Enhancement of associative long-term potentiation by activation of beta-adrenergic receptors at CA1 synapses in rat hippocampal slices.,” *J. Neurosci.*, vol. 23, no. 10, pp. 4173–4181, 2003.

[135] J.-C. Zhang, P.-M. Lau, and G.-Q. Bi, “Gain in sensitivity and loss in temporal contrast of STDP by dopaminergic modulation at hippocampal synapses,” *Proc. Natl. Acad. Sci.*, vol. 106, no. 31, pp. 13028–13033, 2009.

[136] J. Wickens, “Striatal dopamine in motor activation and reward-mediated learning: steps towards a unifying model,” *J. Neural Transm.*, vol. 80, no. 1, pp. 9–31, 1990.

[137] T. Chang, S. H. Jo, K. H. Kim, P. Sheridan, S. Gaba, and W. Lu, “Synaptic behaviors and modeling of a metal oxide memristive device,” *Appl. Phys. A Mater. Sci. Process.*, vol. 102, no. 4, pp. 857–863, 2011.